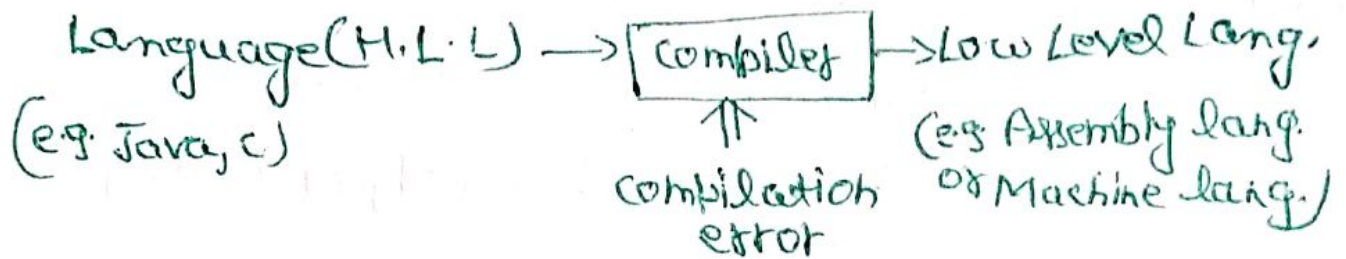
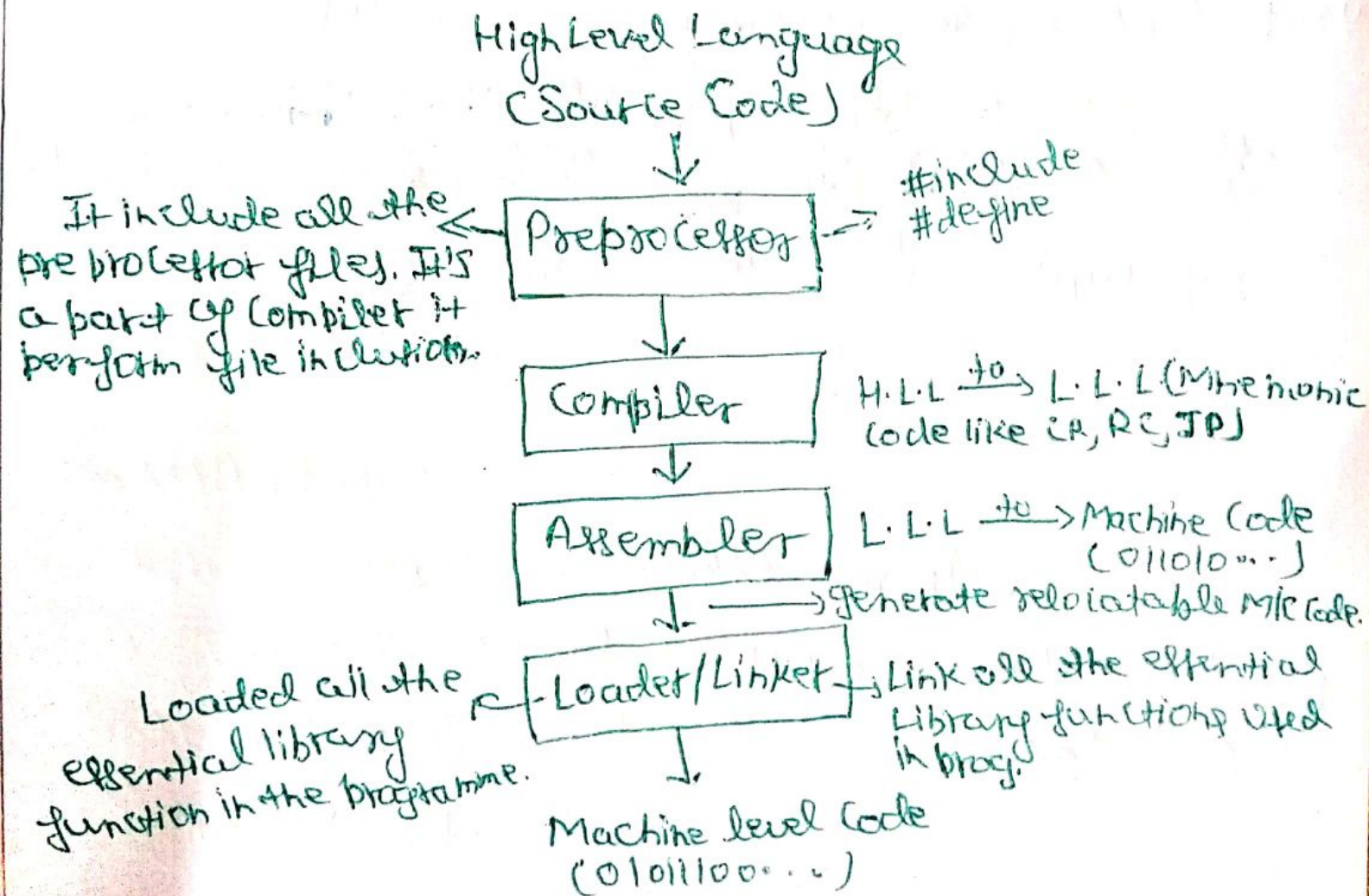


Compiler:-> Compiler is a SW which converts a program written in HLL (High Level Language) to L.L.L. (Low Level Language).



Language processing system



C program {H.L.L.}

②

① C Compiler translates prog. into Low Level language {Assembly language}

⇓

② Assembler translate Low Level language into Machine Code (Object Code)

⇓

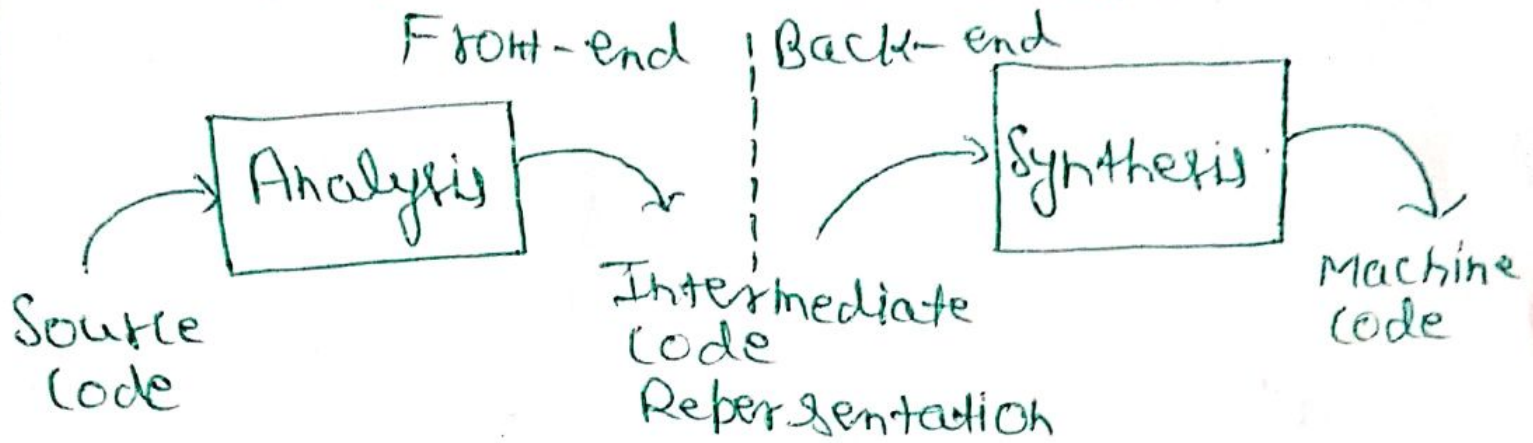
③ Link used to Link all ~~parts~~ parts of prog. together for execution (Executable Machine code)

⇓

④ Loader loads all them into memory & then the prog. is executed.

Phases Of a Compiler

Compiler divided into two phases



Analysis phase :->

- ① Known as Front-end of compiler
- ② Read source prog., divide into code parts & check for Lexical, grammar & Syntax errors.
- ③ Generate intermediate representation of the Source program & symbol table. (Input for Synthesis phase).

Synthesis phase :->

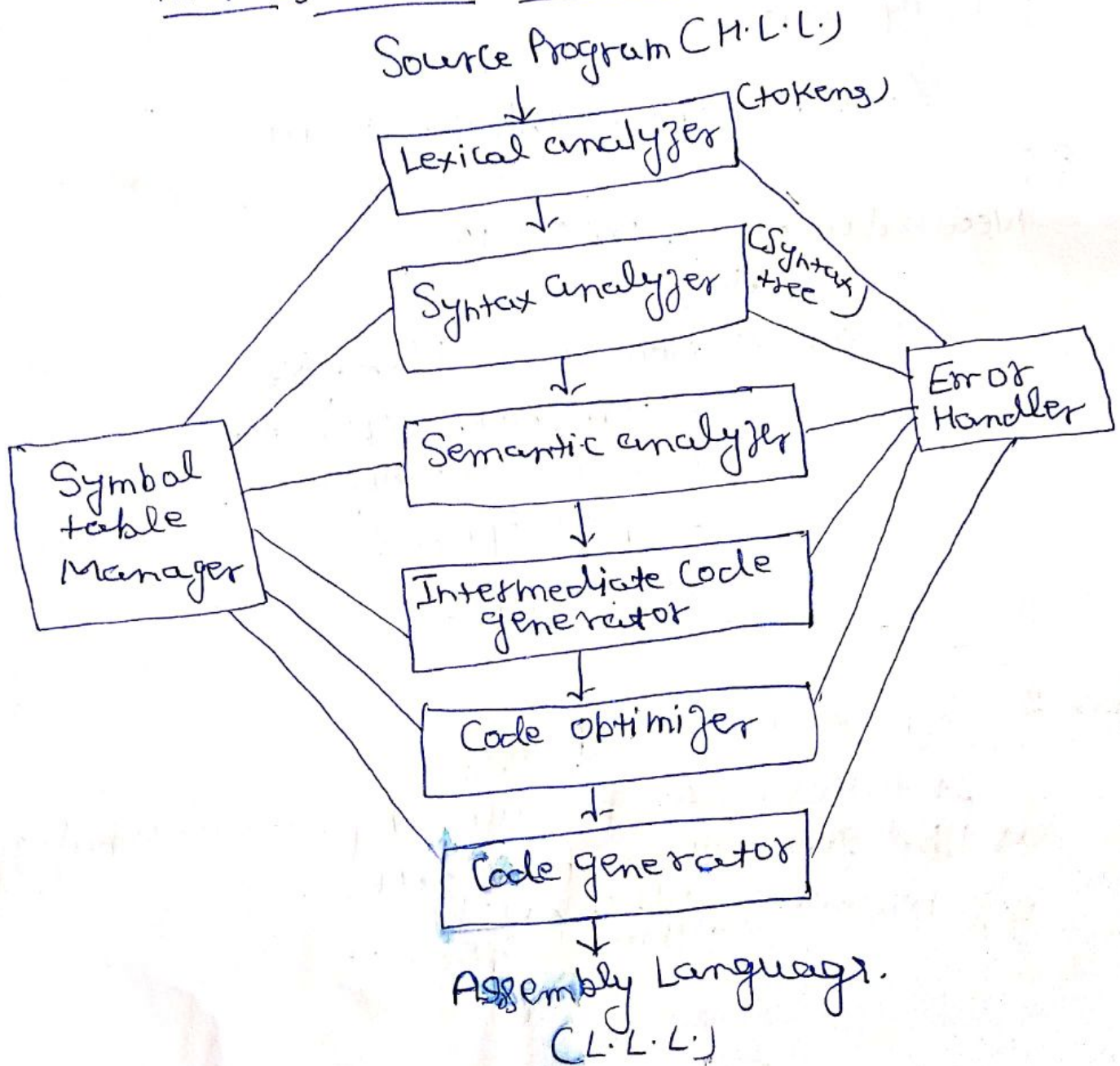
- ① Known as Back-end of the Compiler.
- ② It generate target program with the help of intermediate source code representation & Symbol table.

Phases of Compiler

⑤

Phase:- is a logically interrelated operation that takes source program in one representation & produce output in another representation

There is 6 phases in a Compiler



Phase 1: → Lexical Analysis:-

Reads the stream of characters making up the source prog. & group of char. into sequences called lexeme.

→ Lexical analyzer represent these lexems in the form of tokens

<token-name, attribute-values>

e.g.

New value := old val: + 12

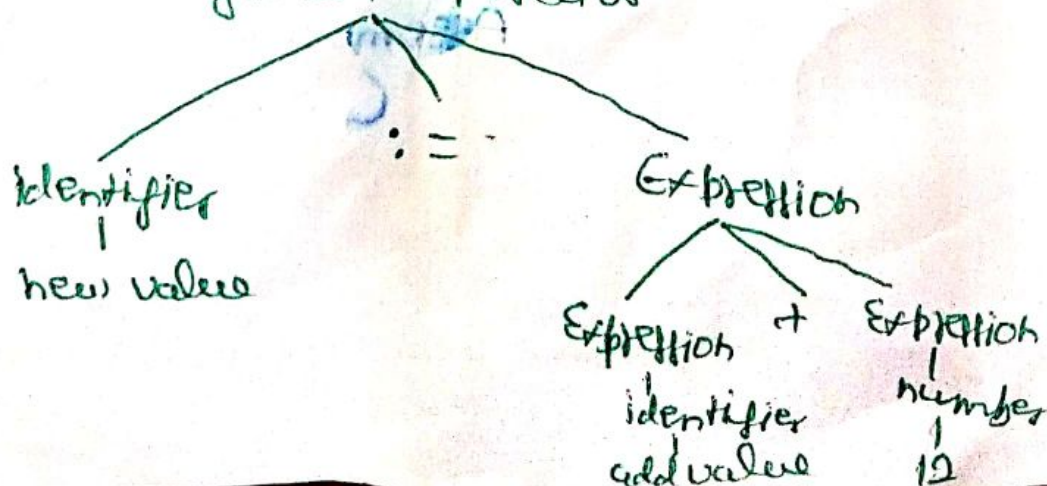
↓

Tokens → New value identifier
= Assignment operator
old val identifier
+ Add Operator
12 Number

Phase 2: → Syntax analyzer (parsing)

It takes token produced by lexical analyzer as input & generates a parse tree.

e.g. new value := (add val + 12)



Phase 3 → Syntax analyzer

Get check whether the parse tree constructed follows the rule of lang.

e.g. assignment of values is b/w compatible data types.

→ also keep track of identifiers, their type & expressions.

e.g. $Sum = a + b^x$

int a, ~~double~~ double Sum;
→ Char b;

data types mismatch.

Semantic records
a: int
Sum: double
b = Char

Note: Phase 1, Phase 2, Phase 3 are synthesis phase.

Phase 4: - Intermediate Code Generation lang.

→ Representation of final Machine code is produce.

→ This is the bridge the analysis & synthesis phase of translation.

e.g. $new\ val. := Old\ val. + fact * i$

⇓
 $id1 := id2 + id3 * 1$

Temp1 = int rel. C11
Temp2 = $id3 * Temp1$
Temp3 = $id2 + Temp2$

} This is the 'Intermediate Code generator.'

$id1 = Temp3$ ⇒ This is final equation.

Page 5! → Code Optimization

↳ Output runs faster & take less place in memory.

e.g.

Temp1 = id3 * 1

id1 = id2 + Temp1.

new val := old value + fact * 1

id1 id2 Temp1

Phase 6: → Code Generation

↳ Translate the intermediate Code into sequence of relocatable machine code.

e.g. id1 := id2 + id3 * 1

MOV R1, Id3.

MOV R1, *1.

MOV R2, Id2

ADD R1, R2

MOV Id1, R1

Assembly language
(C.L.L.)

Symbol table Manager

- ⇒ Symbol tables are data structures that are used by compilers to hold information about source-program constructs.
- ⇒ It is used to store information about the occurrence of various entities like,
 - Objects, classes, variable names, functions etc.
- It is used in both phase Analysis and Synthesis phase.

(Front end)
(Back end)

→ A Symbol table can either be linear or a hash table.

→ It maintain the entry for each name as

$\langle \text{Symbol name, type, attribute} \rangle$
 e.g. $\langle \text{Static, int, Salary} \rangle$

↓↓
 Symbol table store an entry in this format

{ variable declare is }
 Static int Salary

Uses of Symbol table

- ① Symbol table information used by both Analysis phase & Synthesis phase.
- ② To verify that used identifiers have been defined (declared)
- ③ To Verify that Expressions & assignments are semantically correct - type checking.
- ④ To generate intermediate of target code.

BOOTSTRAPPING

(10)

→ Bootstrapping is widely used in the compilation development.

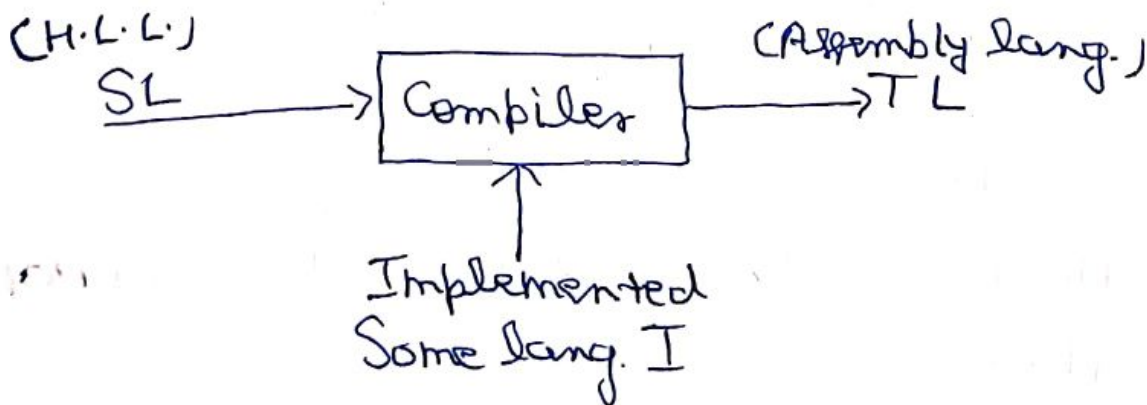
→ It is used to produce a Self-hosting Compiler

It is a type of compiler that can compile its own source code.

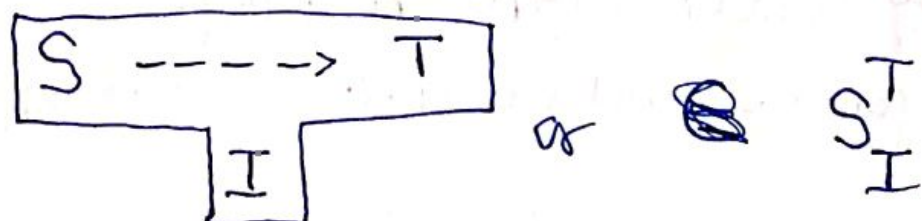
→ A Compiler can be characterized by three languages:-

- ① Source language (S). [H.L.L.]
- ② Target language (T). [Assembly language]
- ③ Implementation language (I).

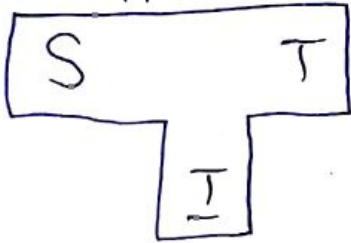
→ Bootstrapping is the process by which Simple lang. is used to translate more complicated program.



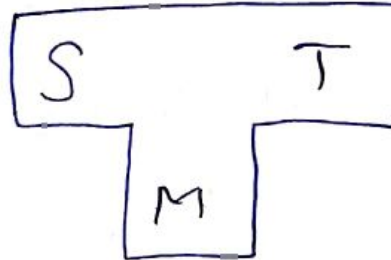
→ Bootstrapping Compiler diagramme is represented by 'T' diagram.



Que i/p



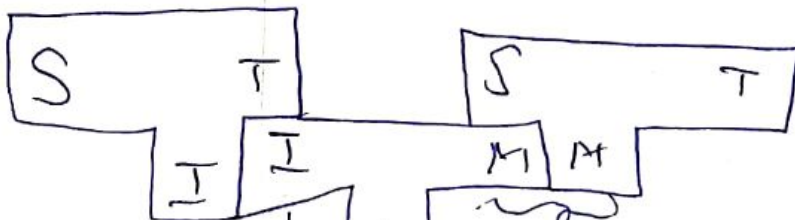
o/p



In b/h
we calculate
by.



Ans.



Must be
Same

Must
be Same

On Same
language.

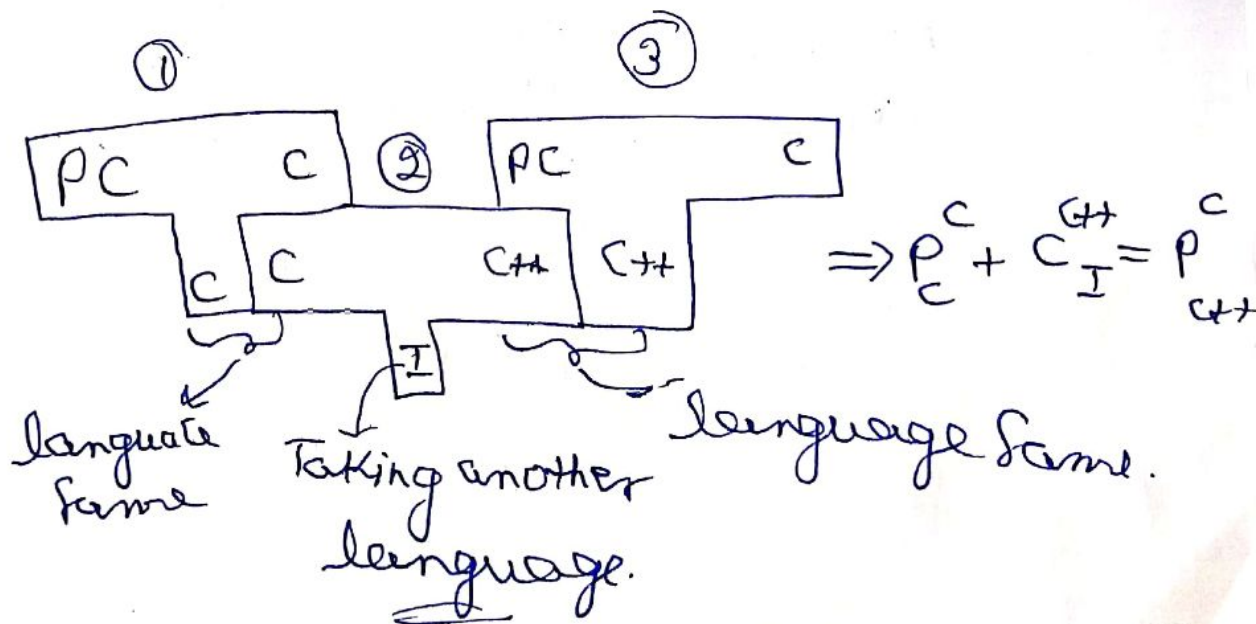
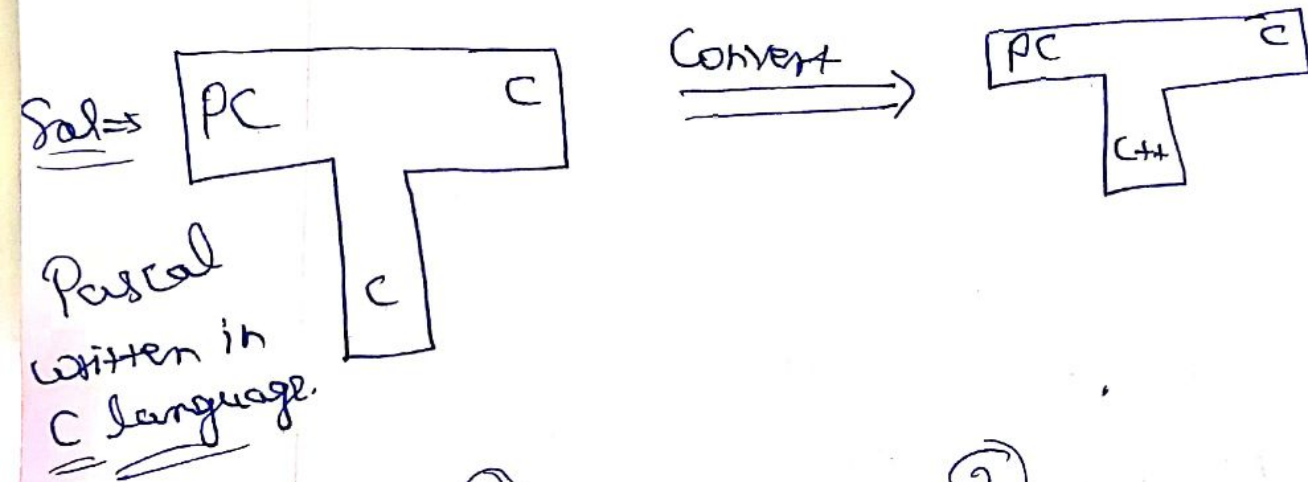
E.g.

Q Pascal Translator written in C lang.

PC — input.

C — output.

Create a pascal translator in C++.



CROSS COMPILER

(13)

A Compiler which runs on one machine but producing Object Code (Target Code) for another machine.

e.g. A Compiler that runs on Windows platform also generates a Code that runs on Linux platform is a Cross Compiler.

⇒ Note: The process of Creating Executable Code for different machines is also called ~~reter~~ "retargeting".

Cross Compiler is also known as "retargetable Compiler".

e.g. G.N.U & GCC are Cross Compilers.

Cross Compiler Operation

