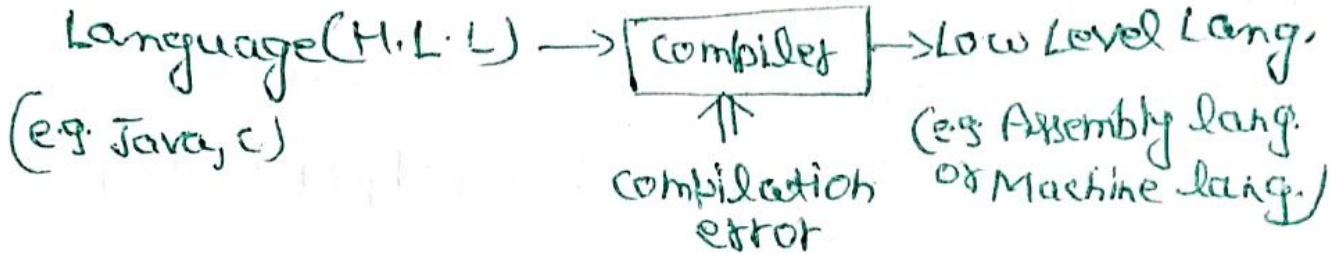
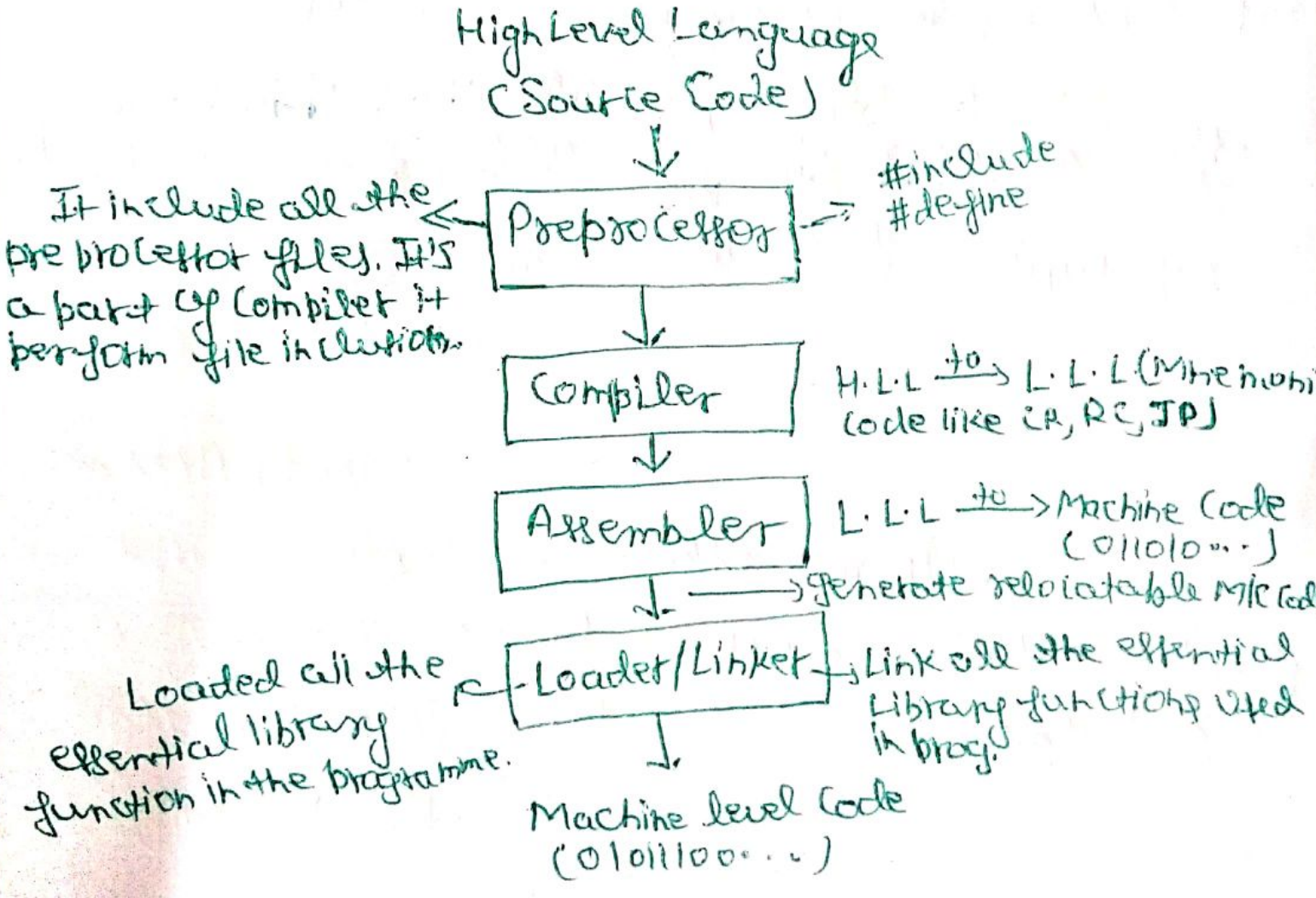
Compiler Design

Compiler:- Compiler is a S/w 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.}

↓

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

↓

② Assembler translate Low Level language into Machine code (object code)

↓

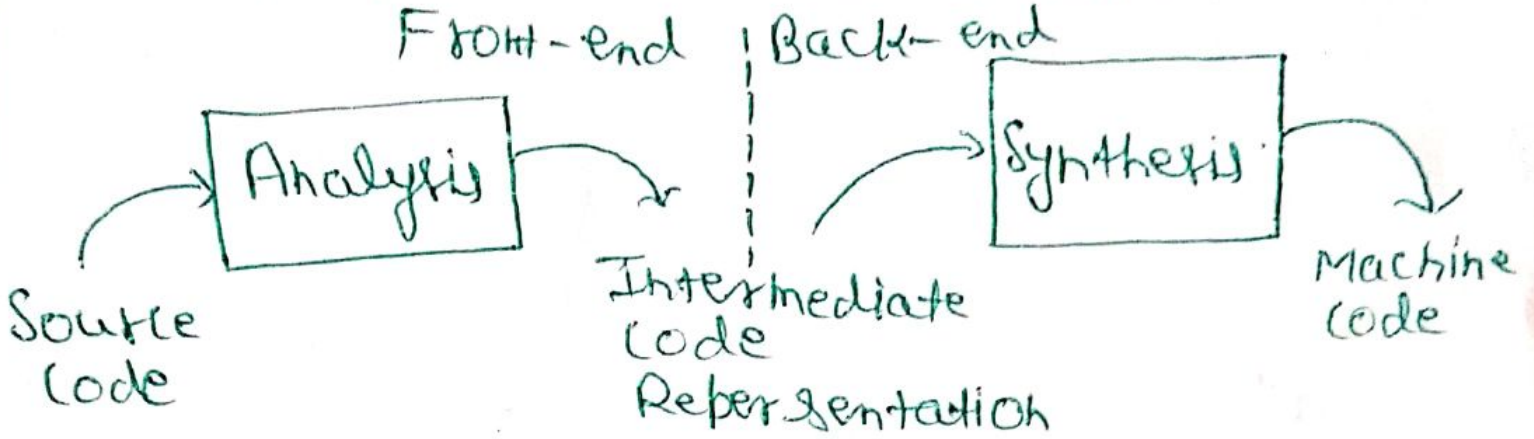
③ Link used to Link all parts of prog. together for execution (executable Machine code)

↓

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 core parts & Check for Lexical, grammar & Syntax errors.
3. Generate intermediate representation of the Source program & symbol table. (Input for Synthesis phase).

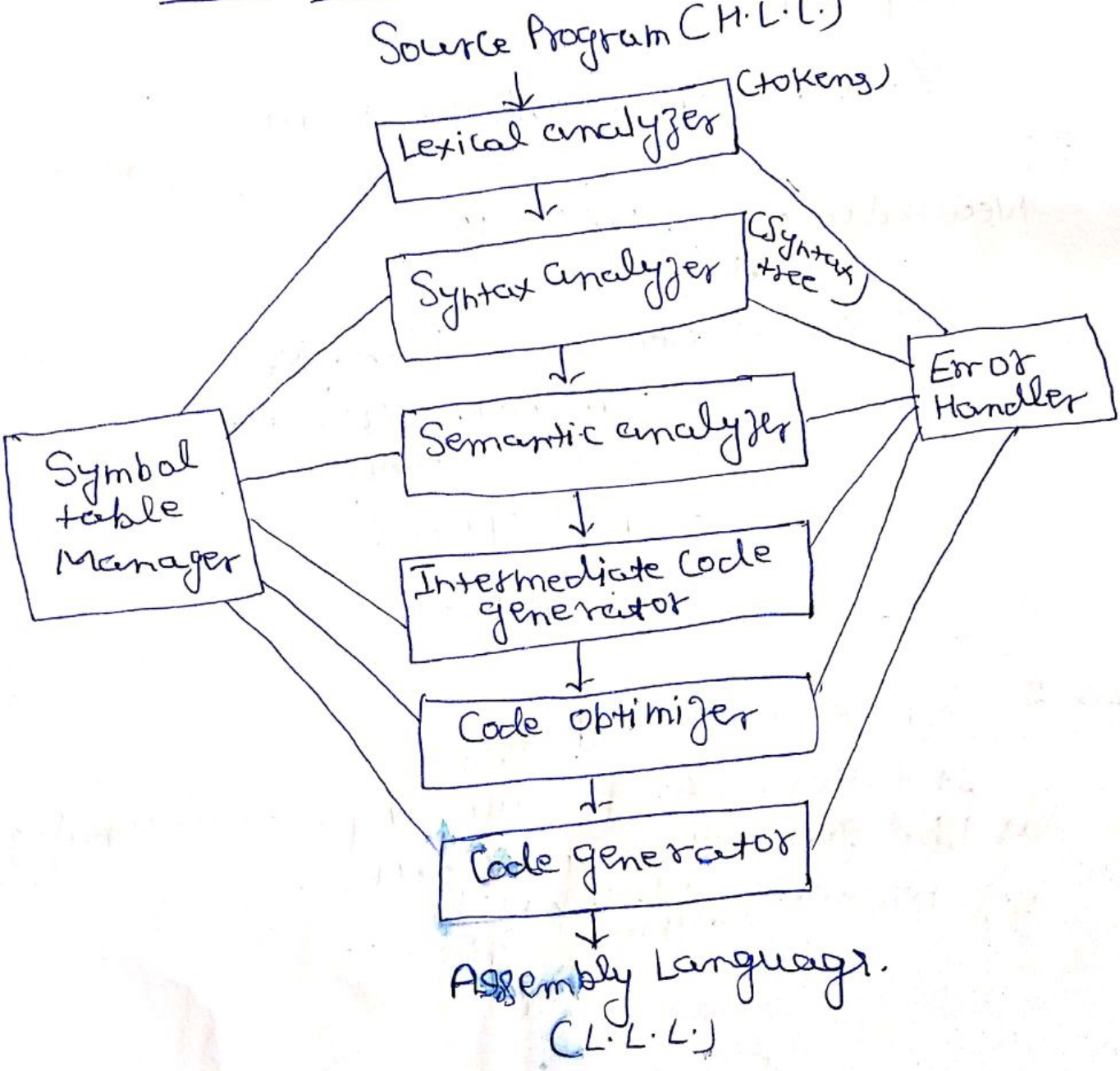
Synthesis phase :->

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

Phase:- is a logically interelated 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-value⟩

e.g.

New value := old val: + 12

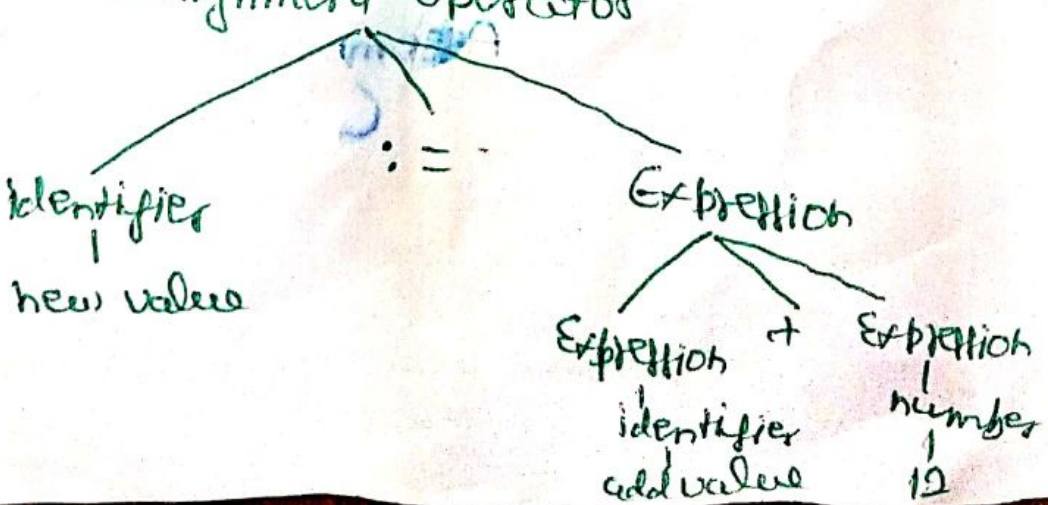
⇓

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 i/p & generates a parse tree.

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



Phase 3 → Syntax analyzer

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

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

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

e.g. Sum = a + b

int a;
~~double~~ char b;
double Sum;

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

→ Representation of final machine code is produce. lang.

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

e.g. new val. := old val + fact * i

⇓

id1 := id2 + id3 * 1

Temp 1 = int val.(1);
Temp 2 = id3 * Temp 1.
Temp 3 = id2 + Temp 2;
id1 = Temp 3 ⇒ This is final equation.

} This is the intermediate code generator.

Phase 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 (L.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 (Front end) and Synthesis phase (Back end).

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

→ It maintain the entry for each name as

<Symbol name, type, attribute>

e.g. <Static, int, Salary>

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

→ 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.

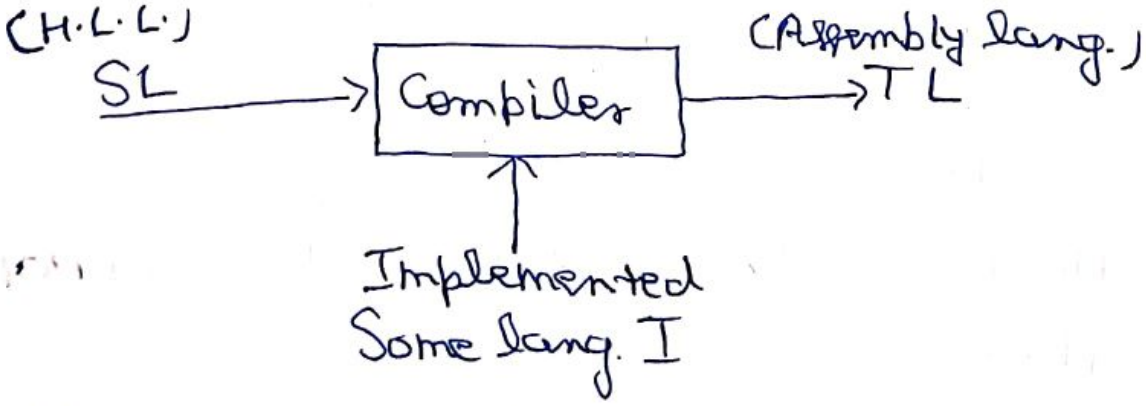
→ A Compiler can be characterized by three languages:-

① Source language (S). [H.L.L]

② Target language (T). [Assembly language]

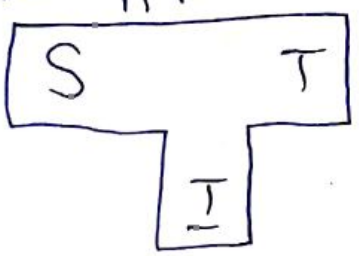
③ Implementation language (I).

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



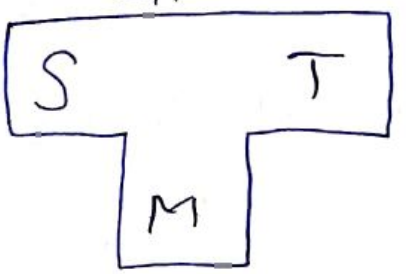
→ Bootstrapping Compiler diagramme is represented by 'T' diagram.

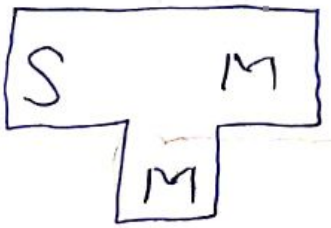
Que i/p



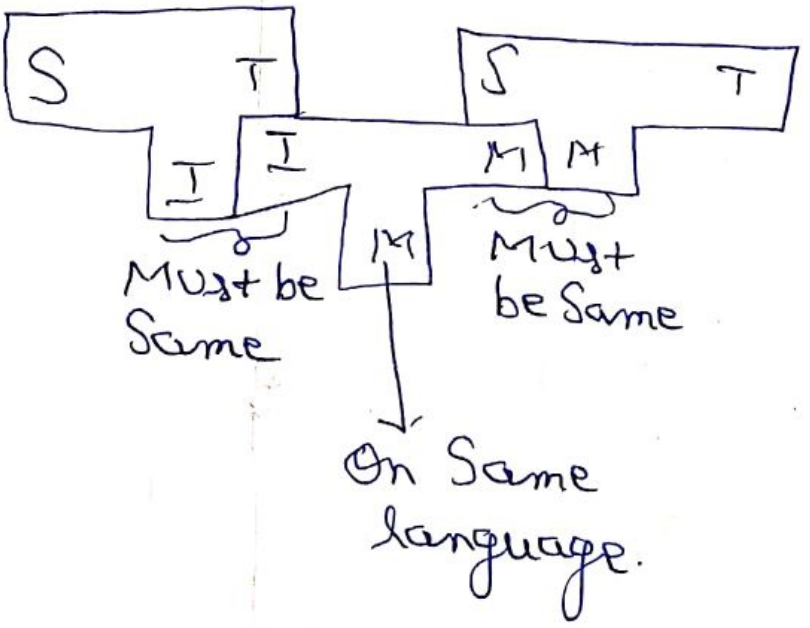
In b/h we calculate by.

o/p





Ans.



E.g.

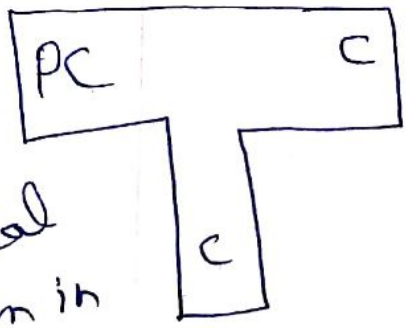
Q) Pascal Translator written in C lang.

PC — input.

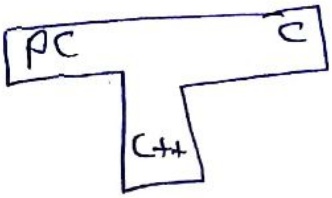
C — output.

Create a pascal translator in C++.

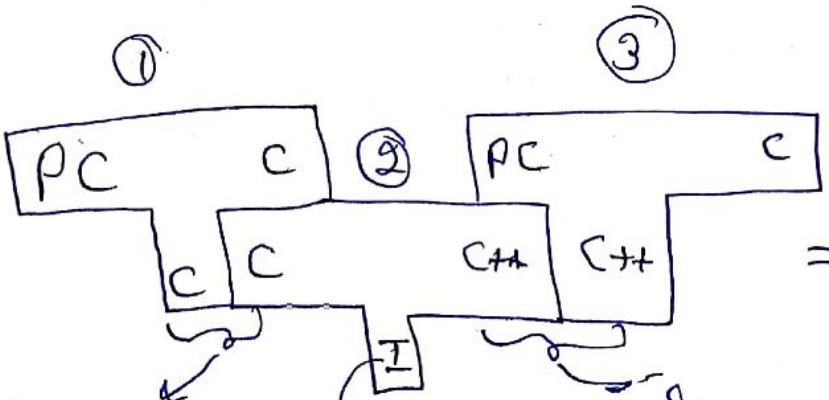
Sol⇒



Convert ⟹



Pascal written in C language.



$\Rightarrow P_C^C + C_I^{C++} = P_{C++}^C$

language same

Taking another language.

language same.

CROSS COMPILER

A Compiler which run on one machine but producing object code (Target Code) for another machine.

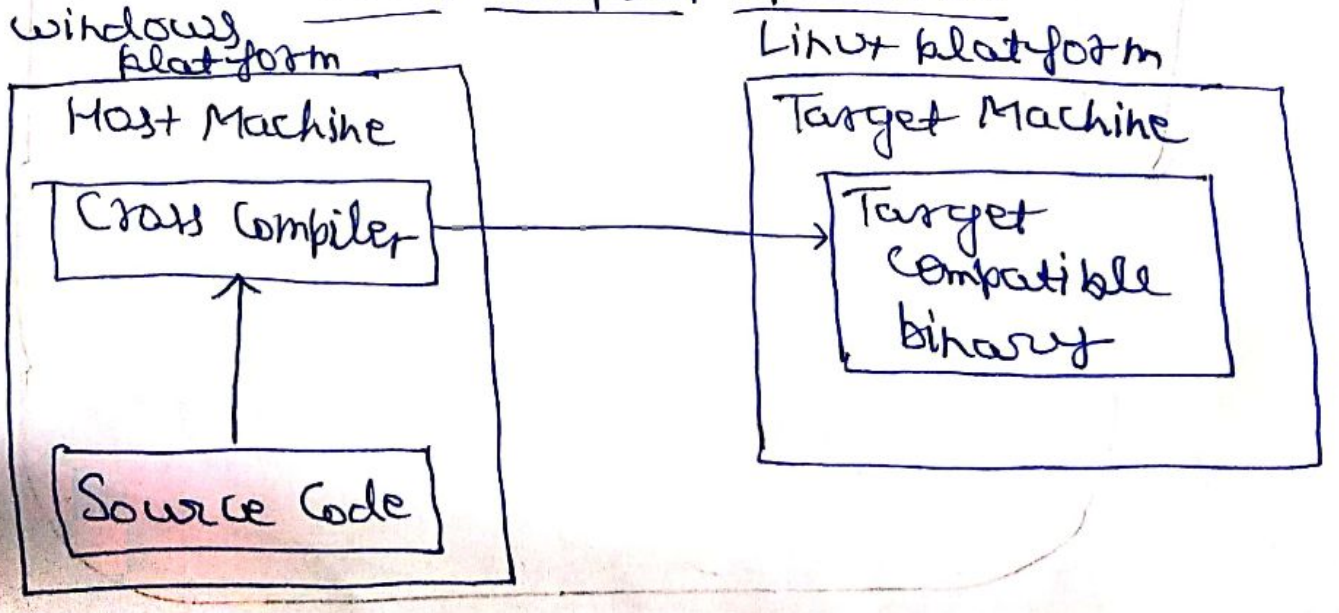
e.g. A Compiler that runs on Window platform also generates a code that runs on Linux platform is a Cross Compiler.

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

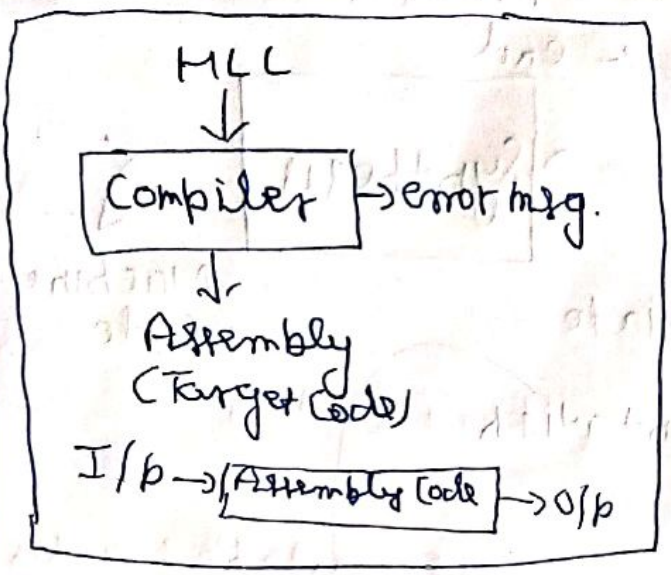
Cross Compiler is also known as "retargetable Compiler"

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

Cross Compiler Operation

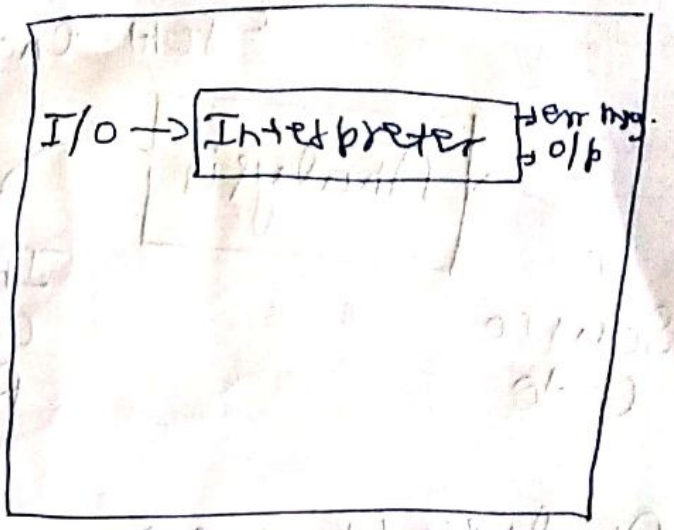


Compilation



1. Translate source code into object code as a whole.
2. It Create object file
3. execution is fast
4. Program not require to translate each time to run.
5. Error diagnosis is poor
6. Most HLL use compiler

Interpreter



Translate the source code one by one & execute immediately It doesn't.

3. execution is slow.
4. Require to translate the program each time to run.
5. Error diagnosis is better
6. few language uses Interpreter.

Finite Automata & Regular Expression Application to Lexical Analyzer.

Lexical Analyzer Process → of recognizing token from Input Source.

Step1:→ L.A. Store the input in input buffer.

Step2:→ The token is read from this input buffer. and Regular Expression are built for a corresponding token.

Step3:→ These R.E. built the F.A. (NON Deterministic F.A.)

Step4:→ Each state of NFA a function is designed & each input along the transitional edges corresponds to i/p parameters of these functions.

Step 5:→ The set of such functions ultimately Create lexical program.

e.g.

Lexical Analysis Process.

```
if(b==0){a=b};
```