Q.1. What is system programming? How is it different from application programming?

→ System programming is the activity of programming computer system software. It is collection of different system programs.

Conceptually, System programming involves designing and writing computer programs that allows the computer hardware to interface with the programmer and user, leading to execution of application software on the computer system.

Application programming →

In application programming any program is designed to perform a specific function directly for the user or in some cases, for another application program.

	System programming	Application programming
①	It is collection of components and art of designing of a given program.	It is set of program that view computer as a tool for solving a particular problem.
②	Programming is done using assembly language which interacts with hardware.	Application programming is used to build application software which include software.
③	System software is a software that executes the application software.	Application software is a software that is been used by end user.

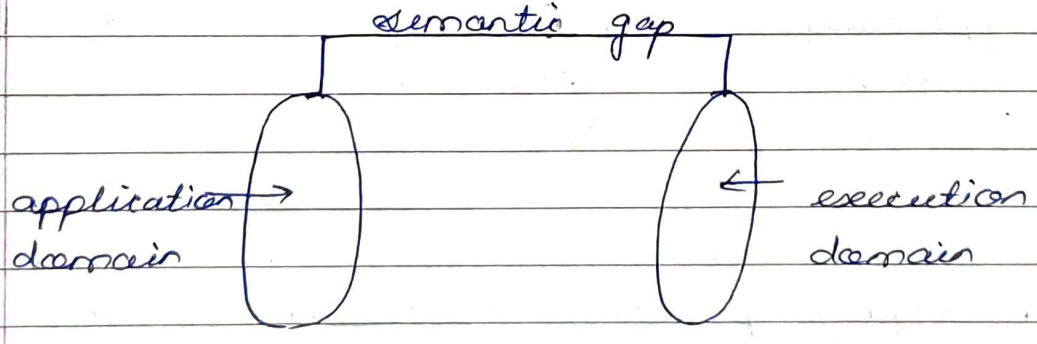
(4)	System programming is used to write low level instr^.	Application programming is used to write high level instr^.
(5)	Ex - loader, linker, compiler.	Ex - calculator, library management system.

Conceptually, computer instr^ or data, anything that can be stored electronically in a system is a system software which helps you to provide better user interaction. It facilitates user to use the hardware system with its maximum efficiency.

Q.2. Define following.

i] Semantic gap.

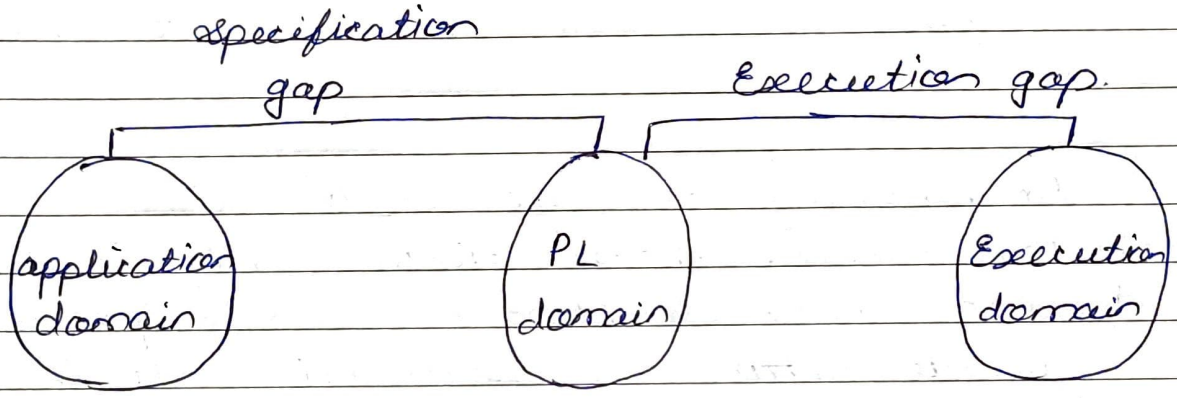
→ The gap between application domain & execution domain is called semantic gap.



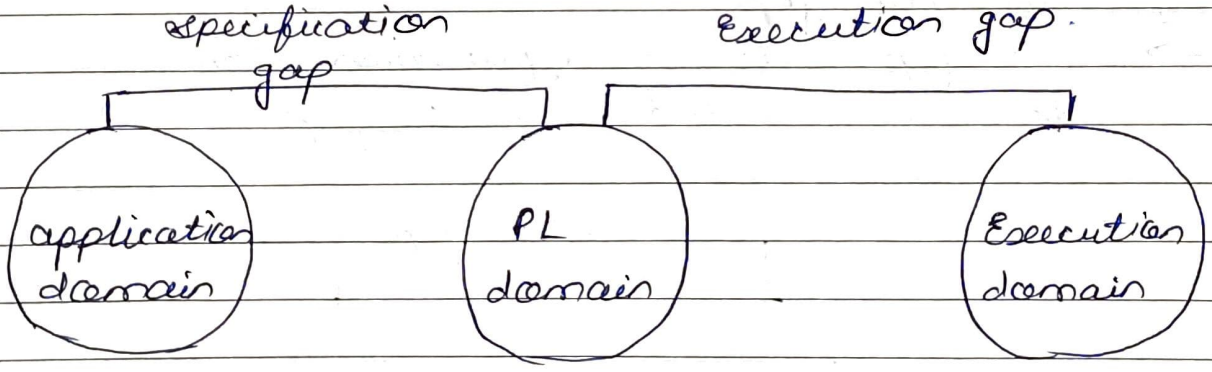
Consequences of semantic gap -

- large development time
- large development efforts
- poor software qualities

2] Execution gap.
It is the gap between the semantics of programs (that perform the same task) written in different programming languages.



3] Specification gap.
It is the semantic gap between two specifications of the same task.



4] Language processor
A language processor is a software that bridges a specification or execution gap.
The program form input to a language processor as the source program and to its output as the target program.
The languages in which these programs are written are called source language and target

prepro
migra

DATE

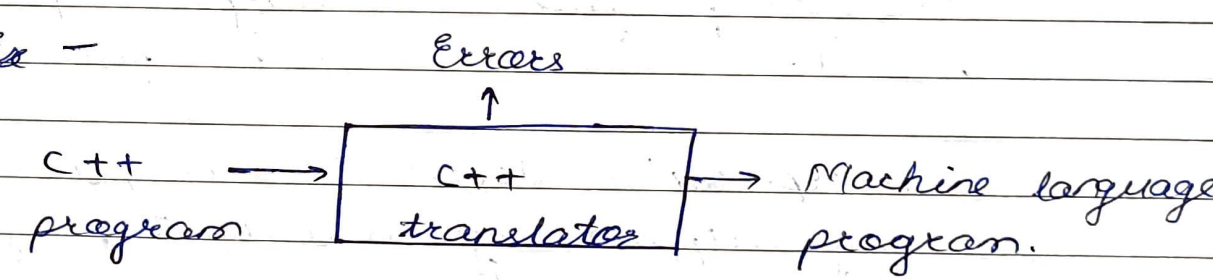
language, esp.

5] language translator.
A language translator bridges an execution gap to machine language (or assembly language) of a computer system.
Ex - assembler, compiler

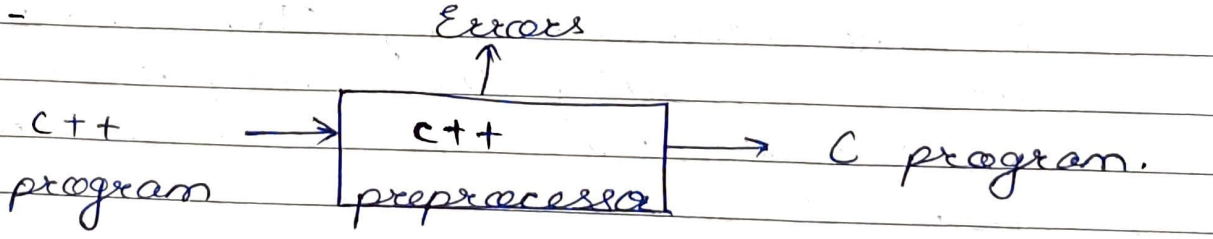
6] language preprocessor.
A language preprocessor is a language processor which bridges an execution gap but is not a language translator.

7] language migrator.
A language migrator bridges the specification gap between two PLs.

→ Ex -



→ Ex -



Q.3. Compare interpreter & compiler.

	Interpreter	Compiler
①	Line by line of program is analyzed in an interpreter.	The entire program is analyzed in a compiler.
②	Machine code is not stored anywhere.	Stores machine code in the disk storage.
③	The execution of the program takes place after every line is evaluated and hence the error is said raised line by line if any.	The execution of program happens only after the entire program is compiled.
④	Interpreted program runs slower.	Compiled program runs faster.
⑤	The interpretation doesn't give any output program & is thus evaluated on every execution.	The compilation gives an output program that runs independently from the source file.
⑥	Due to interpreters being slow in executing the object code, it is preferred less.	Main advantage of compiler is it's execution time.
⑦	It does not convert source code into object code.	It converts the source code into object code.

Ese - Python, Ruby, Perl, SNOBOL, MATLAB	Ese - C, C++, C#, etc.

3.4. Explain in detail fundamentals of language processing activities.

→ - Language processing activities arise due to the differences bet^{n} the manner in which a software designer describes the ideas concerning the behaviour of a software and the manner in which these ideas are implemented in a computer system.

- The designer expresses the ideas in terms of related to the application domain of the software. To implement these ideas, their description has to be interpreted in terms related to execution domain.

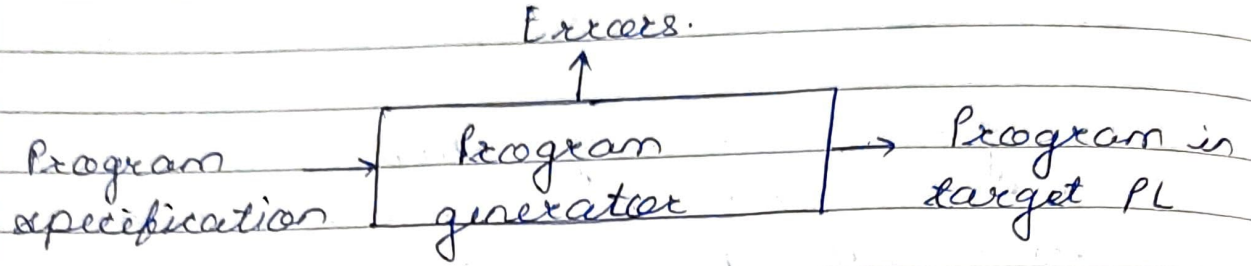
- The fundamentals of language processing activities are divided in two parts:-

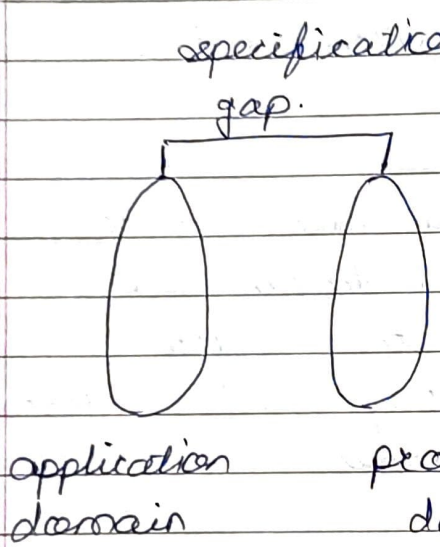
A] Program generation activities -

- The program generator is a software system which accepts the specification of a program to be generated and generates a program in the target PL.

- The specification gap is now the gap between the application domain and program generator domain.
- This gap is smaller than the gap between the application domain and target PL domain.

- Reduction in specification gap increases the reliability of the generated program.



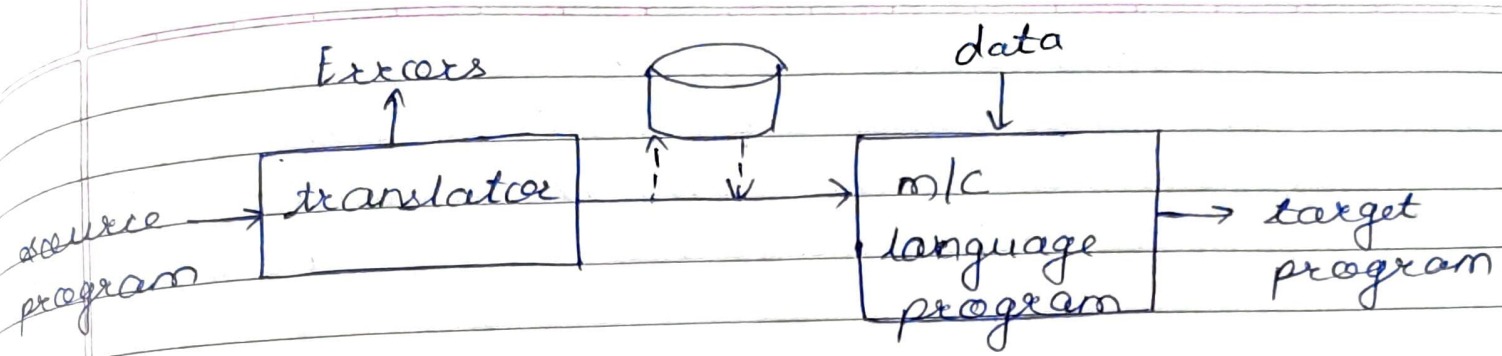


- B] Program execution activities.

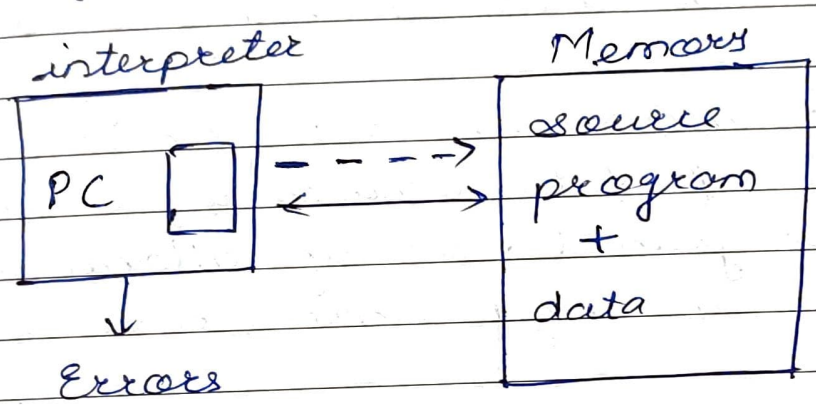
Two popular models for program execution are translation & interpretation.

① Program translation

- A program must be translated before it is executed.
- The translated program may be saved in file. The saved program may be executed repeatedly.



② program interpretation.



Q.5. Explain the fundamentals of language processing.

→ language processing = analysis of SP + synthesis of SP

Collection of LP components engaged in analysis of source program as the analysis phase and components engaged in synthesizing a target program constitute the synthesis phase.

Analysis phase :-
The specification consists of three components :-
① lexical rules - which govern the formation of valid lexical units in the source language.

② syntax rules - which govern the formation of valid statements in the source language.
③ semantic rules - which associate meaning with valid statements of the language.

B) Synthesis phase :-
The synthesis phase is concerned with the construction of target language statements which have the same meaning as a source statement.
It performs two main activities :-
① creation of data structures in the target program (memory allocation)
② generation of target code (code generation)

Q. 9. What is grammar? Explain different types of grammar.

→ Grammar -
It is a finite set of formal rules for generating syntactically correct sentences or meaningful correct sentences.

Grammar is divided in four types :-
① Type-0 grammar
- also called as phrase structure grammar.
- $\alpha ::= \beta$, where both can be strings of Ts & NTs.
- But it is not relevant to specification of programming language.

② Type-1 grammar
- also called as context sensitive grammar.

- $\alpha A \beta ::= \alpha \pi \beta$
- But its is not relevant to specification of programming language.

③ Type - 2 grammar
- also called as context free grammar.
- $A ::= \pi$, which can be applied independent of its context.
- CFGs are ideally suited for PL specification.

④ Type - 3 grammar.
- also called as linear or regular grammar.
- $A ::= tB \mid t$ or $A ::= Bt \mid t$
- Nesting of constructs or matching of parentheses cannot be specified using such productions.

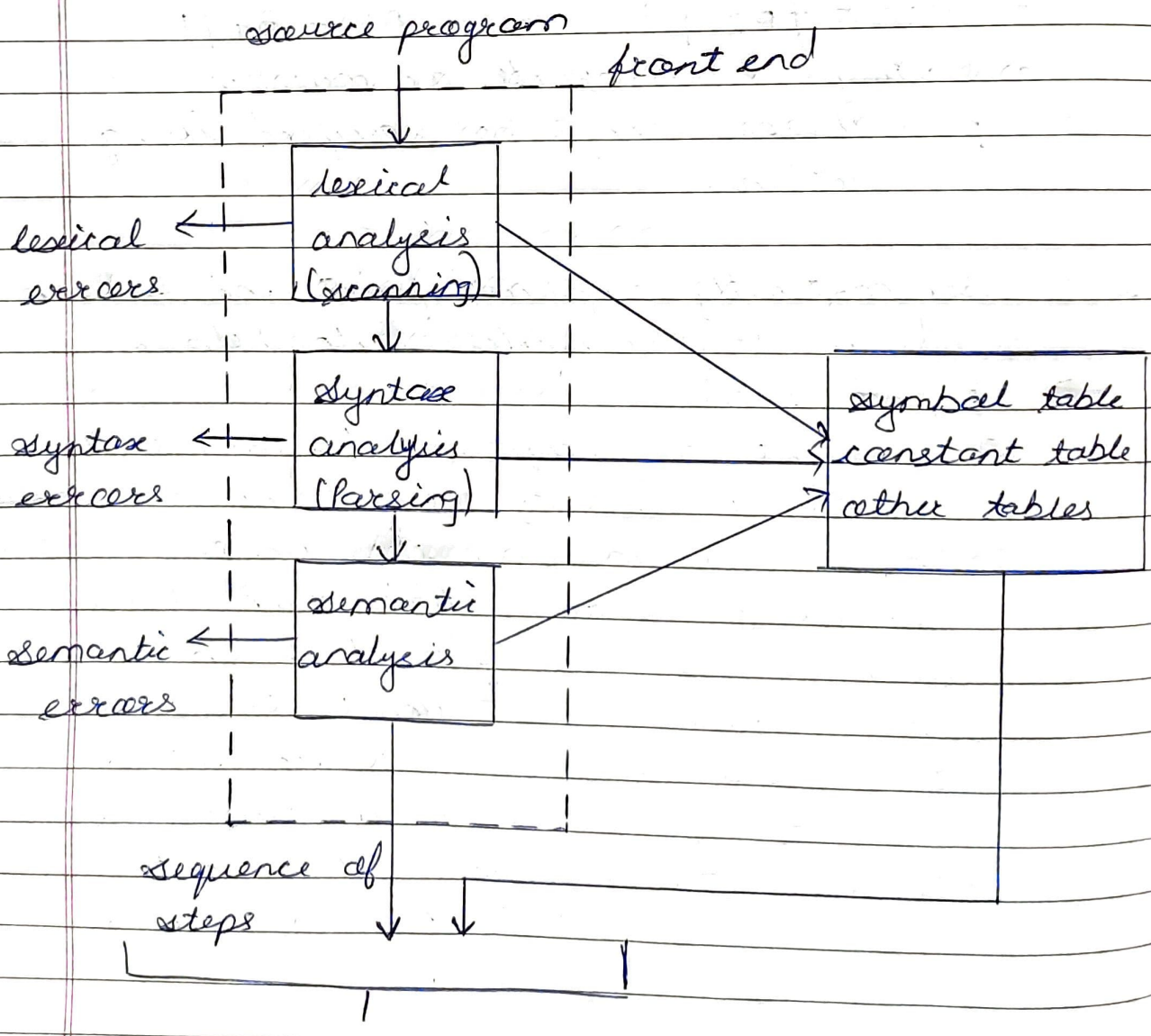
Q.6. Schematically explain front end & back end of toy compiler with help of one.

→ Front end:
1. The front end performs lexical, syntax and semantic analysis of source program.
2. In lexical analysis, the content is the lexical class to which each lexical unit belongs, whether it is an operator, an identifier or a constant.
3. In syntax analysis, a content is semantic structure of a source statement.
4. In semantic analysis, the content is the meaning of statement, for a declaration statement.

5. For simplicity, we use a generic form for the representation of content by various stages of language processor. It consists of two components:-

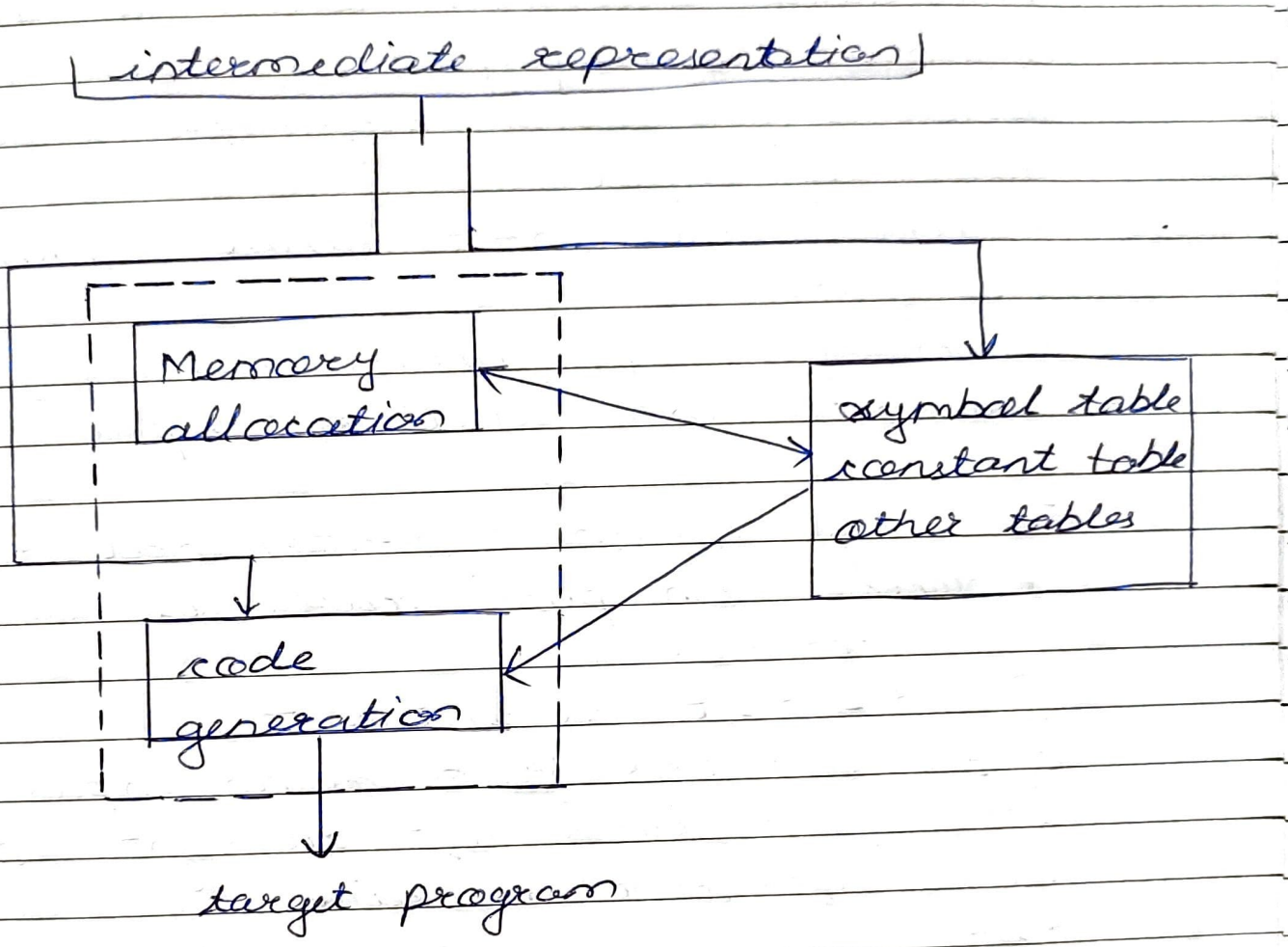
① Tables: The most imp table is the symbol table which contains info about identifiers used in source program.

② Intermediate code: It is sequence of intermediate code units (IC units), where each IC unit is representation of meaningful part of source program.



Back end:

The back end performs memory allocation & code generation.



Memory allocation -
The back end computes memory requirement of a variable from its type, length & dimensionality info found in the symbol name & allocates it to memory.

code generation -
code generation uses knowledge of target architecture.
knowledge of instr^ & addressing mode in target computer, to select appropriate instr^.

Q 8. Explain concept of derivation & reduction with example.

→ Derivation -

A derivation is basically a sequence of production rules, in order to get input string. During parsing, we take two decisions for some ~~sequential~~ sentential form of input:

- deciding the non-terminal which is to be placed.
- deciding the production rule, by which the non-terminal will be replaced.

To decide which non-terminal to be replaced with prod^n rule, we can have two options.

① Left-most derivation :-

If the sentential form of an input is scanned and replaced from left to right, it is called left-most derivation.

Ex -

input string : id + id * id

left-most derivation is

$E \rightarrow E * E$

$E \rightarrow E + E * E$

$E \rightarrow id + E * E$

$E \rightarrow id + id * E$

$E \rightarrow id + id * id$

② Right-most derivation -

If we scan & replace the input with production rules, from right to left, it

is known as right-most derivation.
Eg - input string : id + id * id.
Right-most derivation is :

$E \rightarrow E + E$

$E \rightarrow E + E * E$

$E \rightarrow E + E * id$

$E \rightarrow E + id * id$

$E \rightarrow id + id * id.$

Reduction →
It is the process of replacement of string or part of string by non terminal according to the production rule.

Q.7. What is IR? Prepare an IR by making front end analysis of following problem

```
P, Q : integer
R : read
R := P * Q;
```

→ Intermediate Representation :-

An intermediate representation is a representation of a program between the source & target languages. A good IR is one that is fairly independent of the source & target languages, so that it maximizes its ability to be used in retargetable compiler.

Q. 10. What is binding & binding times? Discuss different binding times & give its imp.

→ Binding -
In computer programming, binding is the mat an association between two or more programming objects or value items for some scope of time & place.

Binding times -
The binding of a program element to a specific characteristic or property is the choice of the property from a set of possible properties. The time during program organization or processing when this choice is made is defined as binding time of that property.

Types of binding times -
1] Execution time (run-time) -
Some bindings are ~~made~~ implemented durin program execution. These contain bindings of variables to their values and the binding of variables to specific storage areas.
The property of runtime bindings are as follows -
① On entry to a subprogram or block -
Binding are limited to appear only at the time of entry to a subprogram or block during execution.

② at arbitary points during execution -
Some bindings can appear at any ~~time~~ point during execution of program.

2] Translation time (compile-time) -

(1) Bindings have been chosen by the programmer -
In writing a program, the programmer consciously produces some decisions concerning choice of variables names, types for variables, program statement structures, etc. that describe bindings during translation.

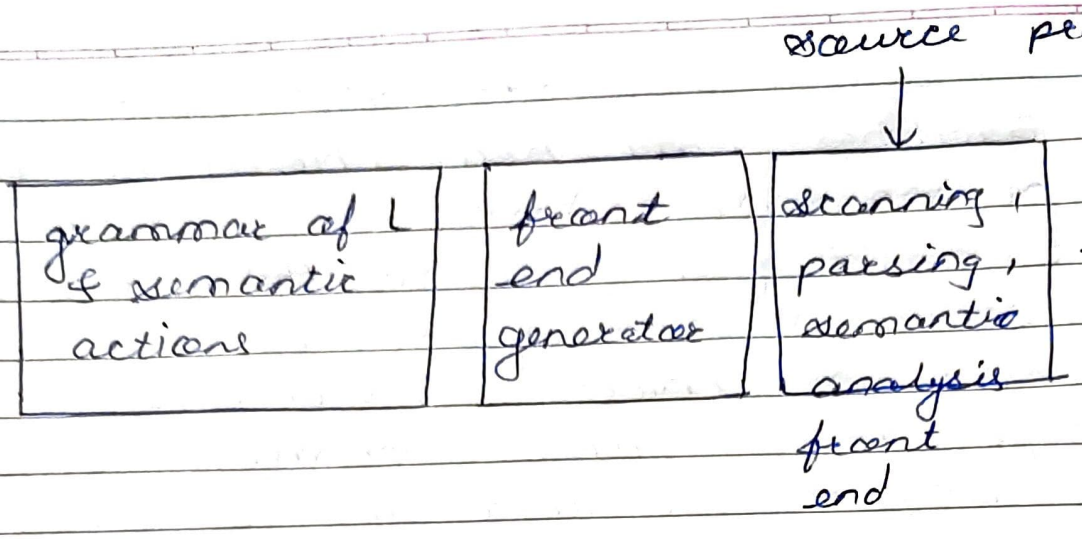
(2) Bindings chosen by the translator.
Some bindings are selected by the language translator without a direct programmer requirement.

(3) Bindings chosen by loader -

Q.11. What is LPDT? Explain LEX & YACC in detail.

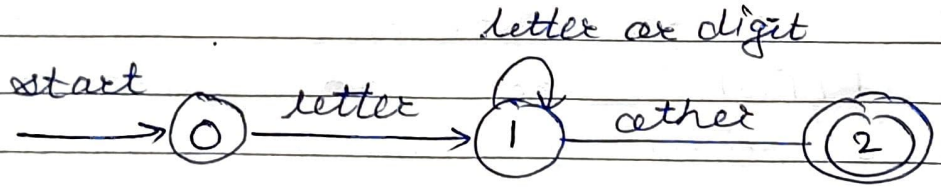
LPDT :-

(1) Set of language processor development tools (LPDTs) focus on generation of analysis phase of language processors.
(2) It generates programs that perform lexical, syntax & ~~syntax~~ semantic analysis of the source program & construct IR.
(3) These programs collectively form the analysis phase of language processor.



LEX

① Lex is a program (generator) that generates lexical analyzers.
② It is mostly used with Yacc parser generator.
③ It reads the input stream & outputs source code implementing the lexical analyzer in C programming language.
④ a simple pattern : letter (letter | digit)*




```
start : goto state0
state0 : read c
         if c = letter goto state1
         goto state0

state1 : read c
         if c = letter goto state1
```

if c = digit goto state 1
goto state 2

state 2 : accept string.

⑤ Lex is good at pattern matching.

Metacharacter	Matches
.	any character except newline
\n	newline
*	zero or more copies of preceding exp
+	one or more copies of preceding exp
?	zero or one copy of preceding exp
^	beginning of line
$	end of line
a|b	a or b
(ab)+	one or more copies of ab
"a+b"	literal "a+b"
[]	character class

YACC

① YACC reads the grammar & generates C code for a parser.
② This is known as bottom-up or shift-reduce parsing.
③ Input to yacc is divided into three sections.

a) ... definitions ...
%. %
The definition section consists of:
• token declarations
• C code bracketed by "%. {" and "%. }".

b) ... rules ...
%. %
The rules section consists of:
BNF grammar

c) ... subroutines ...
The subroutines section consist of user subroutines.