REASONS FOR STUDYING CONCEPTS OF PROGRAMMING LANGUAGE

① Increased capacity to express ideas :

* Depth at which people can think is influenced by the expressive power of the language in which they communicate their thoughts.

  ↳ examples could be from literature

* Complexity in thoughts require deftness in natural languages.

  → It is difficult for people to conceptualize structures that they cannot express/describe verbally or in writing.

② Improved background for choosing appropriate language :

• Familiarity with a wider range of languages and language constructs will enable a programmer to choose the language with features that best address the problem.

③ Increased ability to learn new languages :

- Process of learning new programming language is often lengthy and difficult process.

  ↳ for someone who is comfortable with only one or two languages and has never examined programming language concepts in general

- Learning the concept of programming language facilitates one to learn languages following those concepts.

  ↳ For instance, understanding the concepts of object-oriented programming helps to learn Ruby in a much shorter time.

④ Better understanding of the significance of implementations :

- An understanding of implementation issues leads to an understanding of why languages are designed the way they are.

  ↳ leads to the ability to use a language more intelligently.

- In addition, certain program bugs can be found and fixed only by a programmer who is aware of some of the implementation details

⑤ Better use of languages that are already known

- Contemporary programming languages are large and complex.

  ↳ knowing all features of a language often be difficult.

  ↓

  learning the concepts of a programming language helps to know about the unknown and unused part of the programming language.

INFLUENCES ON LANGUAGE DESIGN

Design of programming languages depends on various factors. For instance,

basic architecture of computers have profound influence on the programming language design.

One of the most prevalent architectures is the

von Neumann architecture

$\hookrightarrow$ Mathematician and physicist

⊞ Most of the popular languages of the past 60 years have been designed by following the prevalent

$\boxed{\text{von Neumann architecture}}$

⊞ von Neumann Architecture

In von Neumann architecture, program instructions and the data on which those program instructions operate, are stored in the memory.

$\hookrightarrow$ Until this approach, each computing machine was designed and built for a single predetermined purpose.

As we see, the three main sub-systems in von Neumann architecture are:

1. Central Processing Unit (CPU)
2. Memory
3. input/output interfaces

CPU:

It is considered as the heart of the computing system

Control unit: Determines the order in which instructions should be executed

↳ Governs the flow of information through the system

Controls the retrieval of the proper operands.

Arithmetic logic unit:

Performs all the mathematical and Boolean operations

Registers: These are temporary storages to store and transfer data and instructions being used.

* Registers have faster access time than memory.
↳ often directly connected to CPU

Memory :

Memory is used to store program, program instructions and data.

Random-access memory (RAM)

stores the data and general-purpose programs that the machine executes.

• Temporary

↳ Content can be changed at any certain moment as needed

It is erased when the computer power is turned off.

Read-only memory (ROM)

• Permanent

↳ Stores initial boot up instructions of the machine

Input/Output /interfaces

Computer's memory receive information and send data using the I/O interface.

For instance, i/o interfaces allow computer to communicate with secondary storage such as disk and tape devices

Key points :

- Data and programs are stored in the same memory
- Central Processing Unit (CPU) is separate from the memory

executes instructions

sequence of statements needed to reach a goal

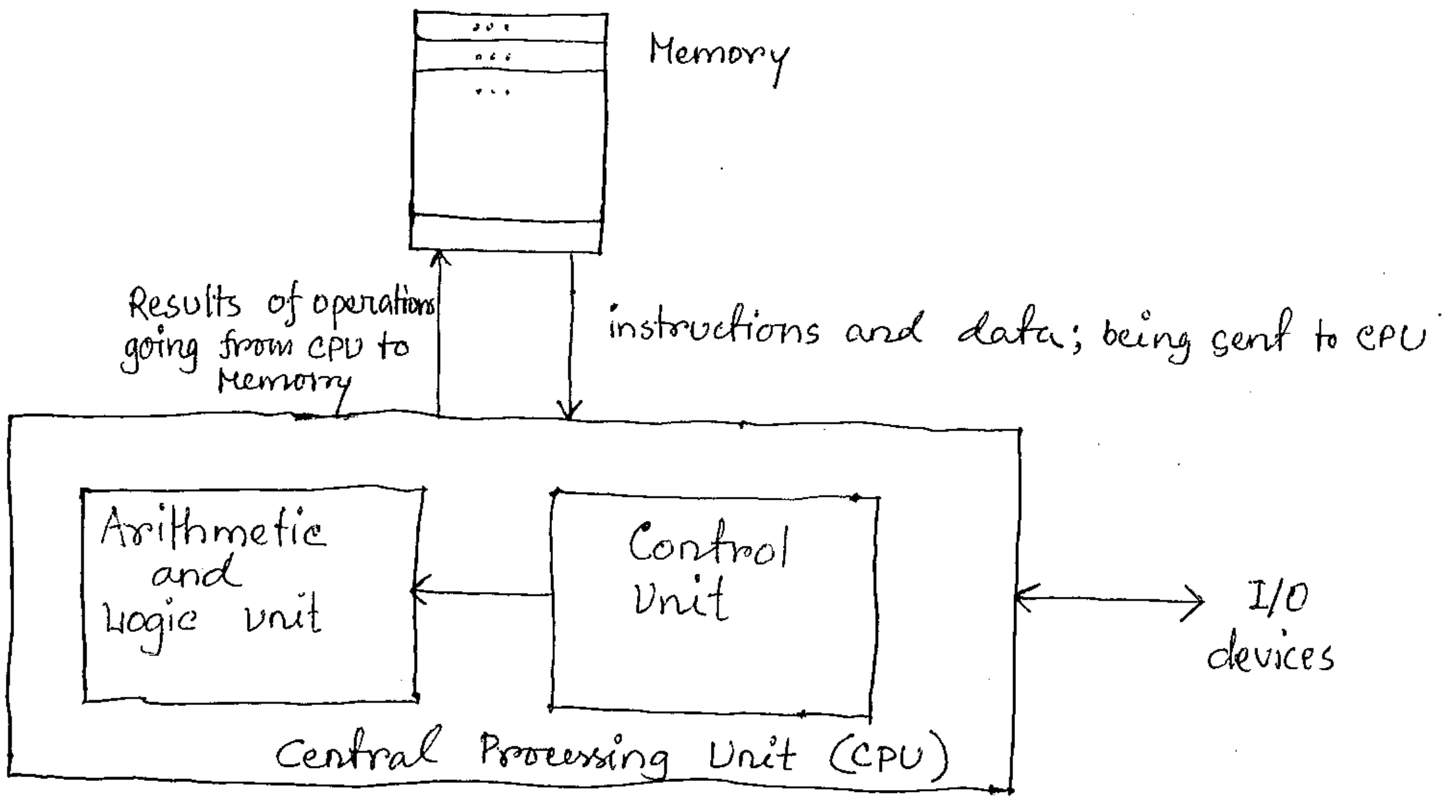
- So, instructions and data must be transmitted to the CPU

↓

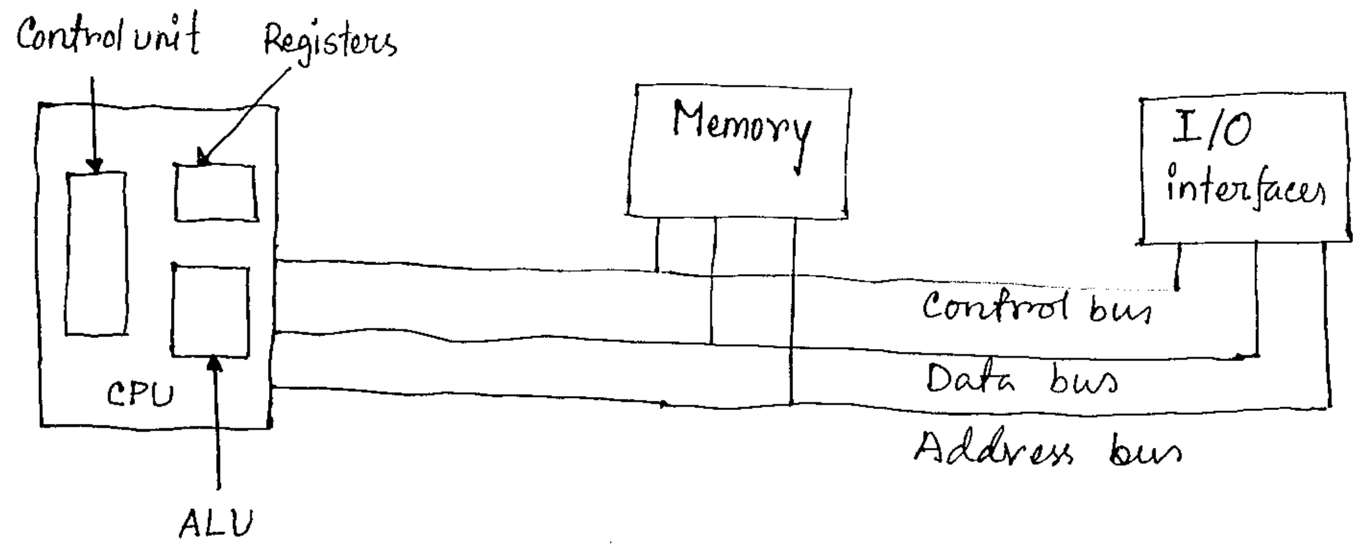
Similar to reach a given/certain goal

Results of operations on data as performed in CPU, are moved back to memory

- Languages designed around the von Neumann architecture is known as Imperative Language


Memory
Results of operations going from CPU to Memory
instructions and data; being sent to CPU
Arithmetic and Logic unit
Control Unit
I/O devices
Central Processing Unit (CPU)

⊞ A more detailed von Neumann architecture is as follows:



⊞ The three main components - CPU, Memory and I/O interfaces - are connected to each other through different types of buses

↳ signal lines

↳ contains several lines/wires that allow for the parallel transmission of information.

Control bus: Consists of signals that permit the CPU to communicate with the memory and I/O devices.

Data bus: A bidirectional bus that sends data to and from a component.

Address bus: It identifies either a memory location or an I/O device.

Machine Language

Around 1940s

von Neumann had the idea that a computer should be permanently hardwired for a small set of general purpose operations.

Given the above, an operator can use a series of <u>binary codes</u> to organize the basic hardware operations in order to solve specific/desired problems.

These binary codes are machine code that a central processing unit (CPU) can directly execute

↳ each instruction, as conveyed using the machine code, instructs CPU to perform specific tasks.
- load
- store
- jump
- Arithmetic Logic Unit operation

Generally,

Machine language is the sequence binary digits that a computer can read and interpret.

It is the only language that a computer is capable of understanding.

Example of Machine language

Let's consider the phrase "Hello World"

$\big\downarrow$ machine language form

01001000 01100101 01101100 01101100 01101111

00100000 01010111 01101111 01110010 01101100

01100100.

A more complex example may be as follows:

Copy a number from a memory location to machine register

register number (registers are high-speed memory cell)

$$\left.\begin{array}{ll} & \underbrace{0010}_{}\,\underbrace{0001}_{}\,000000100 \\ & 0010010000000100 \\ & 0001011001000010 \\ & 0011011000000011 \\ \text{HALT} & 1111000000100101 \end{array}\right\}\ \text{instructions}$$

$$\left.\begin{array}{l} 0000000000000101 \\ 0000000000000110 \\ 0000000000000000 \end{array}\right\}\ \text{Data}$$

Machine codes are all in binary

The above instructions have the below format

$$\underbrace{XXXX}_{\substack{\text{4 bits}\\ \text{opcode}}}\ \underbrace{XXXX\ \ XXXX\ \ XXXX}_{\text{12 bits}}$$

Type of operations to be performed

codes for the instruction's operands.

COMPILATION

Programming languages can be implemented in one of the following three ways:

a. Programs are translated into machine language. This can be executed directly on the computer. Compiler

b. Pure interpretation Programs are interpreted by another program known as Interpreter.

c. The third way of programming language implementation is a compromise between compilers and Interpreters.

↳ Hybrid implementations

⊞ How do computers understand different programming languages ?

Computers only understand machine language and hence, computer programs written in other programming languages are translated into machine language format.

Compiler translates programming language to machine language.

That's why we compile a program written in other languages such as C, C++, Java etc.

Facts : Form of machine language may differ in different operating system.

How a compiler translates a given program into machine language is often decided by the operating system.

◫ Compiler

A translator that produces an equivalent form of the original program suitable for executions.

Two-step process —

Original program is input to the compiler.

↓

A $\boxed{\text{new program}}$ is output from the compiler.

→ Target program

↳ if it is suitable for direct execution, it is executed

if not executable

↳ If not, we need furthere steps:

Generally, the target language is assembly language

So, the target program needs to be translated and an Assembler does this job.

↳ Translates a target program into an object program

Then, linked with other object programs

Loaded into appropriate memory locations before executions.

Lexical Analyzer

- Gathers the characters of the source code into <u>lexical units</u>.
  - ↳ identifiers
  - ↳ special words
  - ↳ operators
  - ↳ punctuation symbols.

```
int main ( )
{ // Two variables for two integers
int   p, q;
p = 10;
return 0; }
```

comment is omitted

'int' 'main' '(' ')' '{' 'int' ...,
TOKENS

Syntax analyzer

- Takes lexical units to form the hierarchical structures known as PARSE TREES
  - ↳ represents syntactic structure of the program
  - ↳ If the input string is possible to produce using the syntax tree / parse tree, the input string is in correct syntax.

Consider the grammar

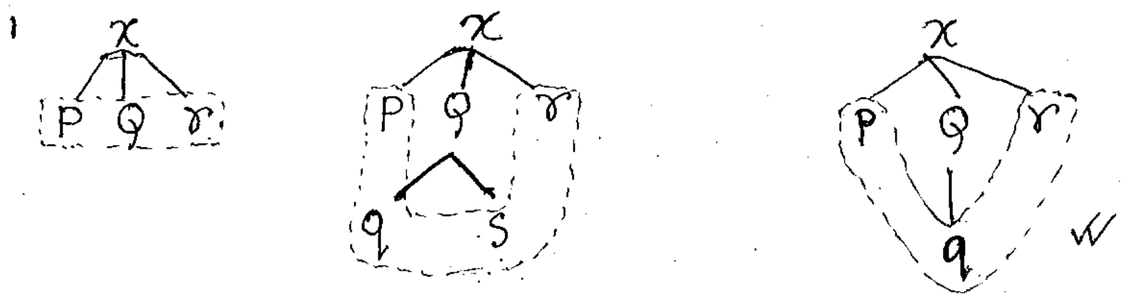
$x \rightarrow pQr$

$Q \rightarrow qS|q$

Let's consider the input string is "pqr"

At the parsing phase, the followings PARSE TREES are formed.

1



Top down Parsing with backtracking

⊞ Intermediator Code generator :

- Produces a program in different language

at an intermediate level between the source program and the final output of the computer.

Sometimes, they are like assembly language or somewhat higher than an assembly language

⊞ Optimization :

- Codes are made smaller and faster or both.
- Generally done on, the intermediate code version

Because, ~~a few to~~ many types of optimization are difficult to do on machine language.

Caution : Optimization process should not delay the overall compiling process.

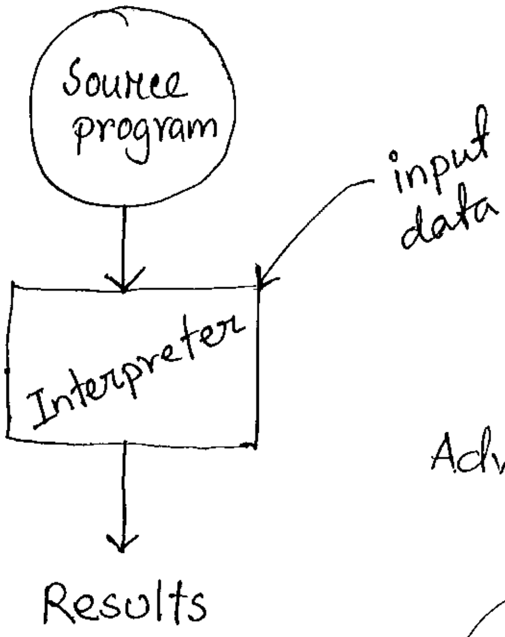
Pure Interpretation

- No translation of the original source code is done.
- Instead, another program is used to interpret the user programs.

Virtual machine

simulates a machine that can perform fetch-execution cycle with the high-level language program statements.

So, the software simulation provides a virtual machine for the language.



Advantages:

- Easy implementation of source-level debugging operations.
- Run-time error messages can refer to source-level units.

For instance,

If array index is out of range, error messages refer to source level units.

Disadvantages

- Execution is about 10 to 100 times slower than compiled system

  ↳ High-level Language statements are complex.

- Regardless of how many times a statement is executed, it must be decoded every time

  ↓ So, rather than connection between CPU and memory statement execution is the bottleneck of a pure interpreter.

- It often requires more space.

  ↳ Because, symbol table must be present during interpretation.

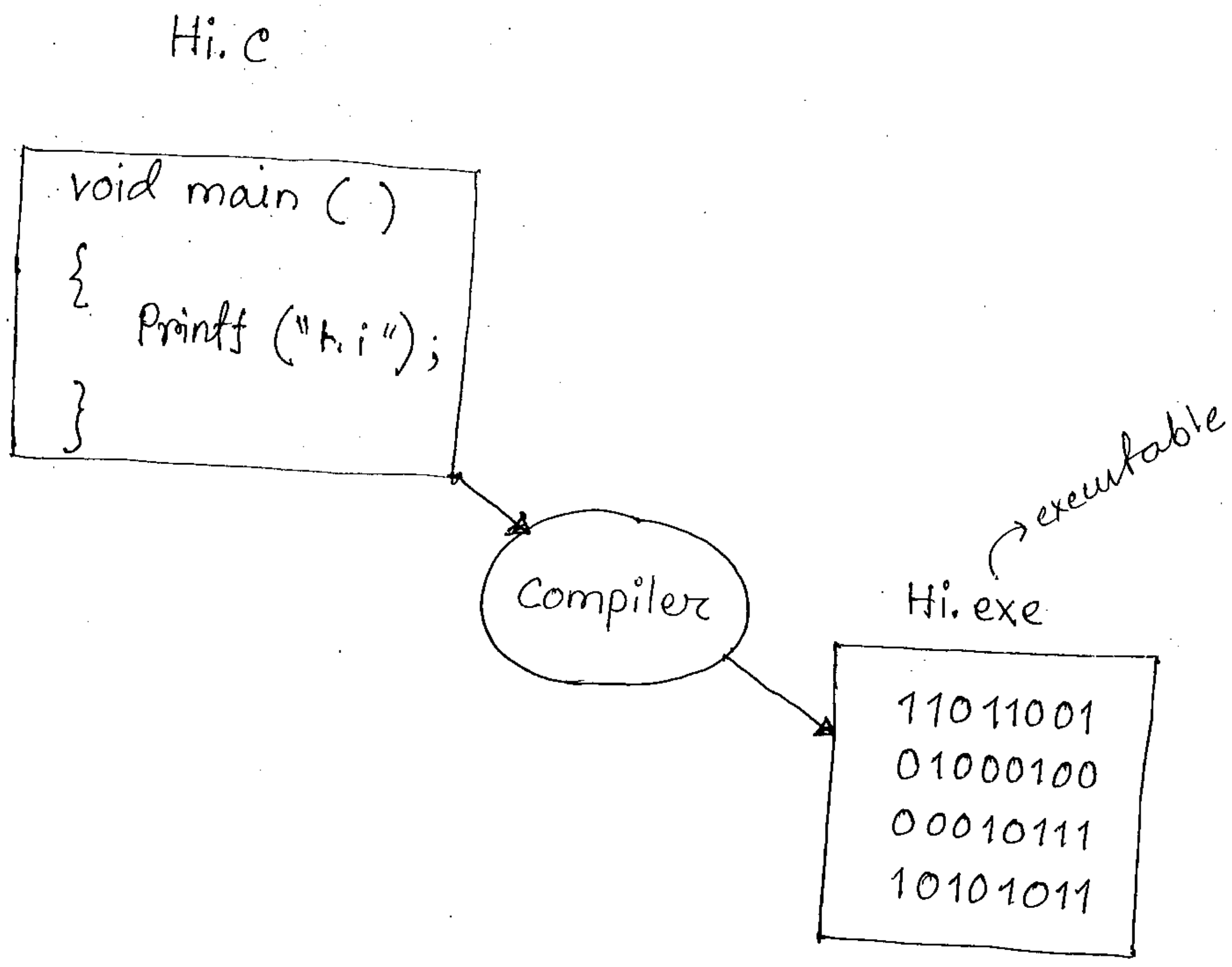
Example: Lisp: List Processing

↳ second-oldest high-level programming language

↳ used widely in Artificial Intelligence research.

Web scripting language { PHP:
JavaScript:

Example



```
void main ( )
{
   Printf ("hi");
}
```

```
11011001
01000100
00010111
10101011
```

⧉ Syntactic Phase Errors :

- Errors in structure
- Missed operators in the source code
- Incorrect spelling
- Incomplete parenthesis

W Increased security for programs; makes it harder to copy.

W Hardware specifications supplied to the compiler may be useful to create a machine code that is optimized for the given hardware specifications.

⧉ Disadvantages :

- Hardware specific
  ↳ should be compiled for 32-bit or 64-bit separately.
- Operating System specific as well — separate versions of Unix and Windows are required
- As optimization of the translated code may need longer time, compil time is high.
- Debugging may be difficult.

⊞ Semantic Analysis and Analyzer

- Checks for errors, for instance type errors, that are often difficult to detect during syntax analysis.

Generally, semantic analysis ensures

Compilation

Advantages:

- Very fast program execution once the translation process is complete.

- It offers better error detection mechanisms.
  - errors are detected at different phases and reported to the user.
  - For instance,
    - Lexical phase errors
    - Syntactic errors
    - Semantic errors

Lexical phase errors:

- Exceeding the length of identifiers, numerical constants etc.
- Presence of inappropriate or illegal characters
- String related errors — mismatched strings

```
printf (" Example of Lexical phase error"); $
```

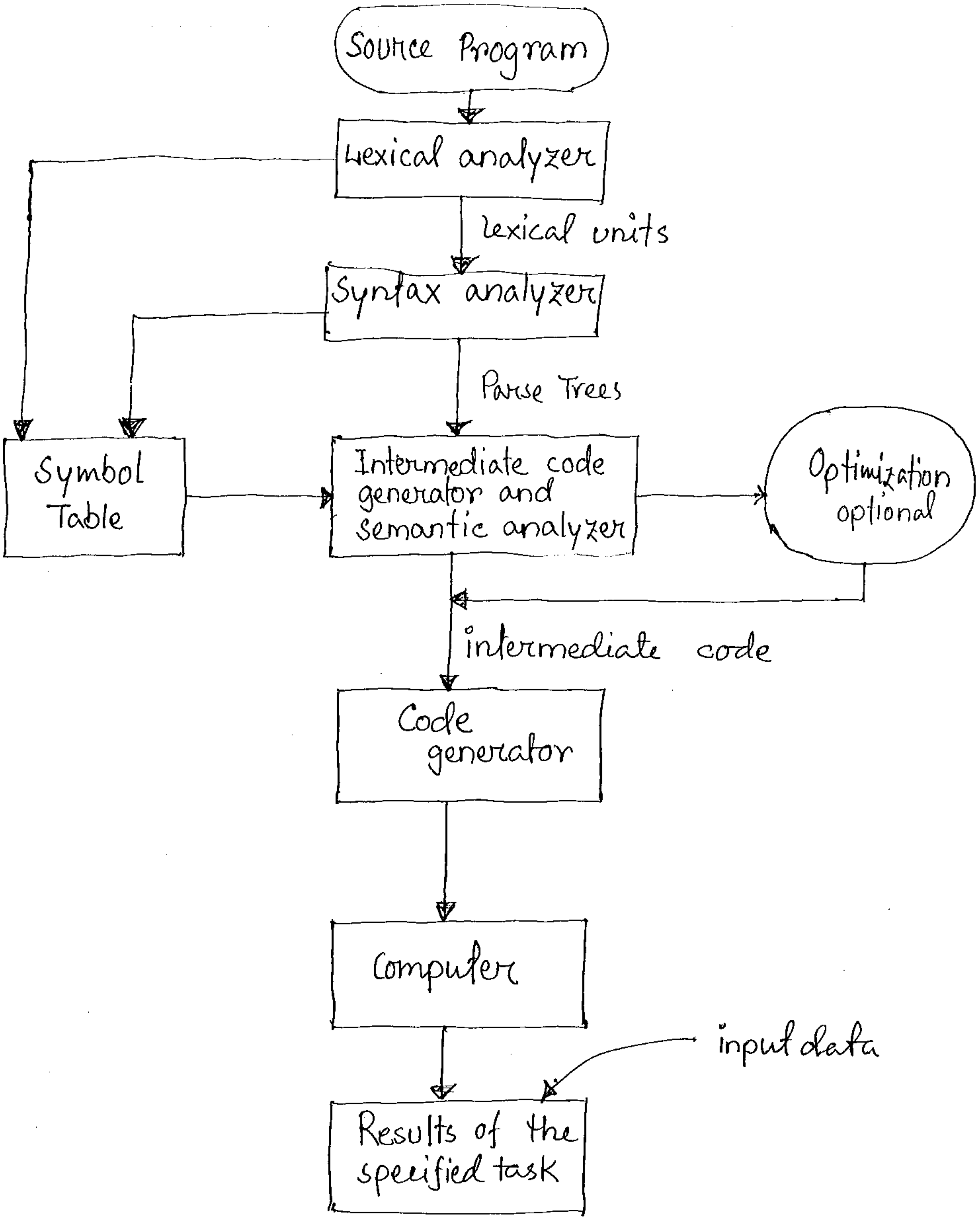
$ → error

⋮

This is an example of a
comment */

beginning of the comment is not added

/*



Flow chart of a compilation Process

田 Additional steps needed for source program execution :

Generally, compiled programs often run along with other programs.

↳ Most user programs also require other programs from operating system

↓

So, before the executable file produced by a compiler can be executed, required programs from operating system must be linked to the user program.

↳ done by Linker

田 User and system code together are sometimes called load module or executable image.

田 The process of collecting systems programs and linking them to user programs is called linking and loading

↳ Achieved by a system program known as Linker

Hybrid Implementation System

- Some language implementation systems translate high-level language programs into an intermediate language form

$$\downarrow \text{which is designed to}$$

Allow easy interpretation

- This <u>approach is faster</u>, than the pure interpretation
  ↳ because, source-code is decoded only once.

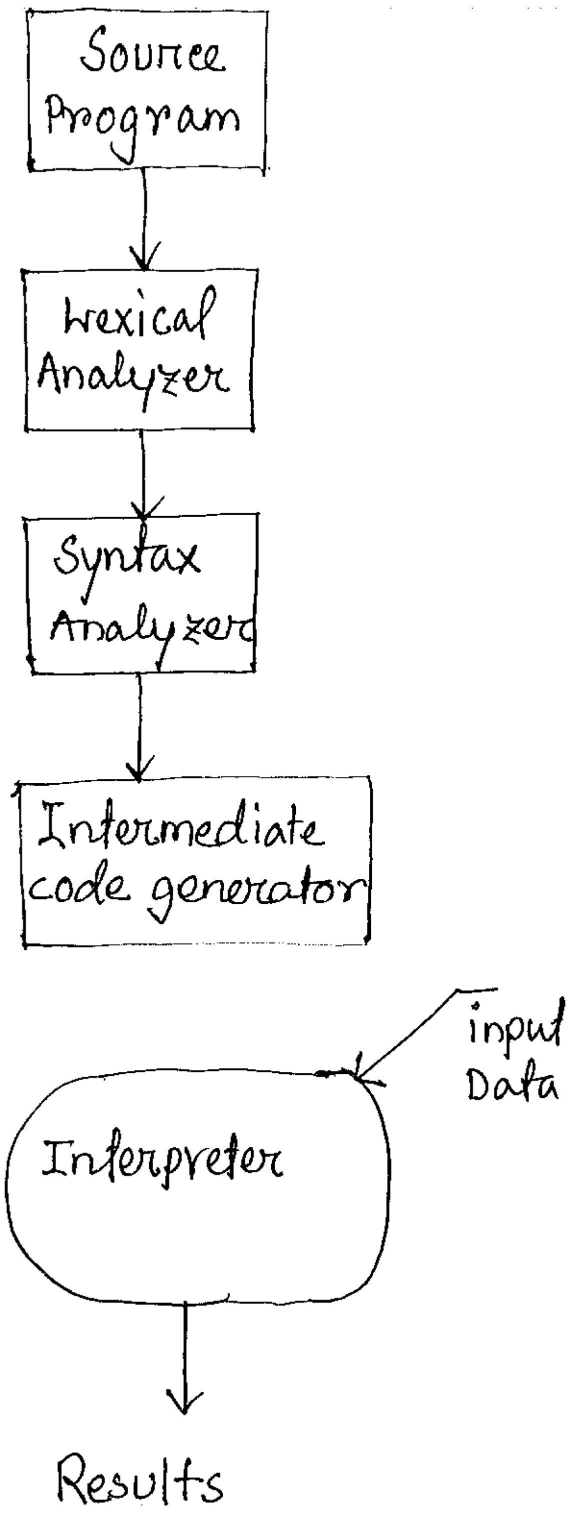
- After the intermediate code is generated, this system does not translate the intermediate code into machine code

$$\downarrow \text{Instead}$$

It simply interprets the inter-mediate code

Example :

HYBRID IMPLEMENTATION SYSTEM



Language Translation

For a programming language to be useful, there must have $\underbrace{\text{translator}}_{\text{a pr gram}}$ that executes a program directly or transforms them into a form suitable for execution.

Translator are of two types —

1. Interpreter
2. Compiler

Interpreter :

A translator that executes program directly is an interpreter.

$$\begin{tikzcd}
 & \text{Source code} \arrow[d] & \\
\text{input} \arrow[r] & \text{Interpreter} \arrow[r] & \text{Output}
\end{tikzcd}$$

Interpretation is a $\underline{\text{one-step process}}$

both the program and the input are provided to the interpreter.

Matlab
Python

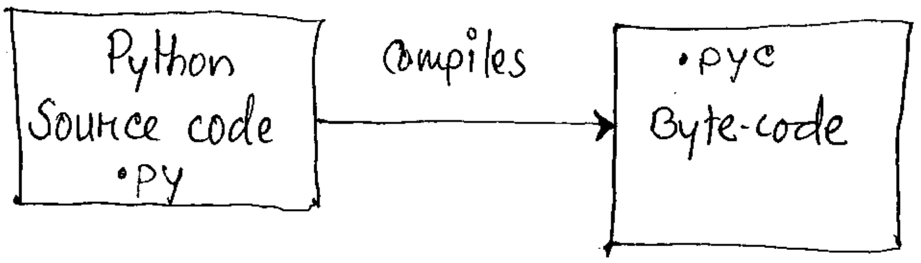
Use interpreters

Byte code:

- A low-leve code that is generated from the source-code through compilation process.

- It can be executed by a virtual machine, or
  can be further compiled into machine code.

Let's consider Python as an example:



Target Language (Byte-code)

*A Form of low-level code known as byte-code

* Compiler translates program's source code to byte-code

Byte-code version of the program is executed by an interpreter,

also, known as virtual machine; written differently for different hardware architecture.

* However, Byte-code is machine-independent

Examples:

Java and Python compile to byte-code and execute on virtual machines.

C and C++ compile to native machine code and execute directly on hardware.

$$\begin{tikzcd}
{} \arrow[r, "\text{source code}"] & \text{Compile} \arrow[r, "\text{Target code}"] & \text{Further Translation} \arrow[r] & \text{Executable code}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{executable code} \arrow[d] & \\
\text{input} \arrow[r] & \text{processor} \arrow[r] & \text{Output}
\end{tikzcd}$$