Computer Fundamentals

Hardware & Software

Hardware	Software
Physical components of a computer. Their core functionalities are input, output, processing and memory.	Set of instructions and data that tells the hardware what processes to do and how to do them.
Tangible (Can be felt)	Intangible (Cannot be felt)
Mouse, Keyboard, CPU, Desktop, etc.	Operating System, Games, Compilers, etc.

Data Hierarchy

The hierarchy of data is:

Bit < Byte < Character < Field < Record < File < Database

- **Bit** → The smallest unit of data, `0` or `1`.
- **Byte** → A group of 8 bits.
- **Character** → A symbol (e.g., `A`, `@`, `5`) stored using one or more bytes.
- **Field** → A set of characters representing one attribute (e.g., Name, Age).
- **Record** → A collection of related fields (e.g., a student's Name, Age, Roll Number).
- **File** → A collection of similar records (e.g., all student records in a class).
- **Database** → A structured collection of files serving a larger purpose (e.g., student marks, attendance, and fees together form a school database).

Hierarchial organisation allows effecient storage, processing and retrieval of data.

Language Levels

There are three levels of computer languages:

- Machine Language
- Assembly Language
- High Level Language

They are similar in terms of what they do, but have very different properties. Let's compare:

	Machine Language	Assembly Language	High Level Language
Syntax	Only `0` and `1`	Mnemonics (`ADD` , `MOV` , `JUMP` etc.)	English-like (`if` , `for` , `print` etc.)
Hardware understanding	Directly understood by CPU	1-to-1 with machine instructions	Abstracted from hardware
Human understanding	Not understandable	Easier than binary, but still technical	Very easy to understand and use
Portability	Not portable (CPU specific)	Not portable (CPU specific)	Portable with compilers
Speed	Fastest execution	Very efficient, close to hardware	Much slower due to abstraction
Control over Hardware	Full control	High control	Limited control
Translation	Not needed - Directly executed by CPU	Assembler → Converts assembly to machine code	Compiler / Interpreter → Converts to assembly or machine code
Uses	Writing device firmware, microcontrollers	System programming, embedded systems	Applications, O. S., AI, wep apps, games

From this, it can be inferred that machine language and high-level language are on either extremes, with machine language being more CPU friendly, while high-level language being more human friendly.

Compilers & Interpreters

Compilers and Interpreters are both translators that translates high-level languages to machine language. But, both of them differ in terms of how they translate. Here is a quick comparison:

	Compiler	Interpreter
Execution	Converts the entire code into machine language before execution	Translates and executes the code line by line.
Output	Creates an executable `.exe` file that can be directly run without compiling	Does not create any such file. Interpreter is needed each time
Runtime speed	Fast (if compiled already)	Slow
Error Reporting	Reports all errors at once during compilation	Reports an error upon finding a faulty line
Portability	Executable files are machine specific	Can be run on any machine with the interpreter
Memory usage	More space (executable file and the source code if necassery)	Less space (executes directly)
Uses	Large, performance critical applications (apps, games)	Scripting, rapid prototyping, education
Example languages	`C` , `C++` , `Java`	`Python` , `Ruby` , `JavaScript`

Now, we know that `C` uses a compiler.

`C` language

Why `C` ?

- This language provides a wide range of operators and keywords
- Minimal but powerful core features
- Mid-level language. Meaning, it is human friendly, yet gives good control over hardware
- Portable. Program can run on different machines with minimal changes
- Compiled into fast machine code, and as efficient as assembly language.

Why `C` is a mid-level language?

- It provides low-level functionalities such as addressing, pointers, reference etc. Hence it is more low-level than most other high-level languages.

- It has high-level constructs. Uses human understandable structure such as functions and modularity. Hence, it is not as low-level as assembly language is.

Overall, we can say that `C` falls between low-level and high-level, or it is a **mid-level** language.

Summary

- Computers comprise **hardware** (physical components for input, output, processing, memory) and **software** (programs and data that direct hardware).
- **Data hierarchy**: Bit < Byte < Character < Field < Record < File < Database. This structure enables efficient storage, processing, and retrieval of data.
- **Language levels**:
  - Machine language: binary, fastest, hardware-specific, hard for humans.
  - Assembly: mnemonics mapped 1-to-1 to machine code, high control, still CPU-specific.
  - High-level: English-like, portable via translators, easier to use but more abstract and typically slower.
- **Compilers vs Interpreters**:
  - Compiler: translates whole program to an executable before running, fast at runtime, reports errors together, machine-specific output.
  - Interpreter: translates and runs line by line, slower at runtime, reports errors as encountered, needs interpreter each run, more portable.
- **C language**: compiled, portable, efficient with rich operators and a minimal core. Considered a **mid-level** language because it offers low-level features (addresses, pointers) along with high-level constructs (functions, modularity).