Questions

3a. What are High level languages and Low-level languages? How are they being different?

3b. Write in detail the features of the C programming language.

3c. Write differences between a compiler and interpreter.

Answers

Ans 3a. High Level Language (HLL) are programming language that are **easy to understand.** They are **Easy to write.** We can **Easily fix** the code of we get errors. The codes are **Portable.** They are **Often platform independent/machine independent. These are primary choices of programming.** Some HLL are Python, Java, C, C++. **E.g. print ("Hello World!!")**

Low Level Language (LLL) are **Difficult to understand.** They are also **Difficult to write.** So, it is very **Difficult to debug** for any person. It is **Not portable** so we can't run the same code on different machine. It is **Machine dependent.** It is **Primary choices for machine programming.** But even after that LLL are **Comparatively fast. E.g.: Assembly Language, Machine Code, ADA, C, Fortran etc. Example of LLL 100100 011100**

Difference between HLL & LLL are as follows. Low level languages are closer to hardware resources and there is very less abstraction layering in them while High level languages have higher level of abstraction which makes low level languages more efficient and have more control over hardware while high level languages have precise control and easy to implement.

Feature	High-Level Language	Low-Level Language
Abstraction	High, abstracts hardware details	Low, close to hardware
Ease of Use	Easy to read, write, and understand	Difficult to read, write, and understand
Syntax	Human-readable, uses words and symbols	Machine or assembly code, uses binary or mnemonic codes
Portability	Highly portable, can run on different systems with minimal changes	Low portability, often specific to a type of hardware
Development Speed	Fast, due to easier syntax and higher abstraction	Slow, requires detailed management of hardware
Memory Management	Automatic (handled by the language)	Manual (handled by the programmer)
Control Over Hardware	Limited control over hardware specifics	High control over hardware specifics
Performance	Generally slower due to abstraction overhead	Generally faster due to direct hardware manipulation
Debugging and Testing	Easier, with advanced tools and error handling	Harder, with fewer tools and more manual effort
Examples	Python, Java, C++, Ruby	Assembly language, machine code

3b. Features of C programming language are:

- **Procedural Language:** C follows a procedural paradigm, where **predefined instructions are executed step by step**. Programs can contain multiple functions to perform specific tasks. Unlike object-oriented languages, C primarily relies on this procedural approach.

- **Fast and Efficient:** C is known for its speed and efficiency. While newer languages like Java and Python offer more features, C's direct hardware manipulation allows it to perform faster. Being statically typed also contributes to its efficiency.

- **Modularity:** C code can be organized into libraries for future reuse. These libraries hold much of C's power, allowing developers to solve common problems efficiently.

- **Statically Typed:** C is a statically typed language, meaning variable types are checked at compile time, not runtime. Programmers must explicitly specify variable types, enhancing code reliability.

- **General-Purpose:** C is versatile and used in various applications: Operating systems (Windows, Linux, iOS, Android, macOS). Databases (PostgreSQL, Oracle, MySQL, MS SQL Server, etc.).
- **Rich Set of Built-in Operators:** C provides a diverse set of built-in operators for writing both complex and simplified programs.
- **Libraries with Rich Functions:** Robust libraries and functions in C simplify coding, even for beginners.
- **Middle-Level Language:** C strikes a balance between low-level memory access and high-level abstractions. This makes it suitable for system programming, such as operating systems and compilers.

In summary, C's simplicity, efficiency, and versatility have made it a foundational language in software development.

Ans 3c. The difference between **Compiler** & **Interpreter** are here as follows.

Compiler creates an intermediate object code. Interpreter directly executed by the interpreter.

Compiler	**Interpreter**
The compiler saves the Machine Language in form of Machine Code on disks.	**The Interpreter does not save the Machine Language.**
Compiled codes run faster than Interpreter.	Interpreted codes run slower than Compiler.
Linking-Loading Model is the basic working model of the Compiler.	The Interpretation Model is the basic working model of the Interpreter.
The compiler generates an output in the form of (.exe).	The interpreter does not generate any output.
Any change in the source program after the compilation requires recompiling the entire code.	Any change in the source program during the translation does not require retranslation of the entire code.
Errors are displayed in Compiler after Compiling together at the current time.	Errors are displayed in every single line.
The compiler can see code upfront which helps in running the code faster because of performing Optimization.	The Interpreter works by line working of Code, that's why Optimization is a little slower compared to Compilers.
It does not require source code for later execution.	It requires source code for later execution.
Execution of the program takes place only after the whole program is compiled.	Execution of the program happens after every line is checked or evaluated.
Compilers more often take a large amount of time for analyzing the source code.	In comparison, Interpreters take less time for analyzing the source code.
CPU utilization is more in the case of a Compiler.	CPU utilization is less in the case of a Interpreter.
The use of Compilers mostly happens in Production Environment.	The use of Interpreters is mostly in Programming & Development Environments.
Object code is permanently saved for future use.	No object code is saved for future use.