1) What are device drivers?

A) A device driver is a special kind of software program that controls a specific hardware device attached to a computer. Device drivers are essential for a computer to work properly.

→ Device drivers are necessary to permit a computer to inteface and interact with specific devices. They define the messages and mechanisms whereby the computer. the OS and applications can access the device or make requests for the device to fulfill. They also handle device responses and messages for delivery to the computer.

Ex:- keyboards, mice, CD/DVD drives, controllers, printers, graphic cards and ports.

2) Differences between general purpose system and Embedded system.

General purpose system	Embedded system
1) System is multipurpose and can be used for variety of applications.	1) System is combination of special purpose hardware and Embedded operating system for specific application.
2) General purpose operating system like windows, Linux, MAC.	2) operating system may not be present, or Real time operating system
3) Functionality can be altered by execution different program and application.	3) Non alterable since the firmware is programmed for specific task
4) key factor is performance and speed	4) key factor is application specific, time.
5) The power consumption is more since it is large & multipurpose system	5) less ~~power~~ consumption because it is application specific system
6) Need not to be deterministic soft Real time	6) Deterministic for certain type of Embedded system
7) larger in size	7) Small in ~~[illegible]~~ size

3) How can hardware understand the code that we write in embedded Systems (.c to .exe)

A) Hardware understands the codes written for Embedded Systems through a process called compilation. These codes written in programming languages such as C or C++ are compiled into machine code, which consists of low-level instructions that the hardware can directly execute.

⇒ To convert a C program to an executable, you can use a C compiler. The most common C compilers are GCC (GNU compiler collection) and Clang. If we are using visual studio for C then when you build and compile visual studio automatically generates .exe application file.

⇒ The compiler takes .c source code file and produces an intermediate file known as an object file. This object files contains machine code that's specific to the target platform but is not yet a standalone executable

4) RTOS and GPOS

RTOS (Real-Time operating system) and GPOS (General-purpose operating system) are two different types of operating systems designed for specific purposes:

1) RTOS:-

⇒ RTOS is designed for applications that require precise and deterministic timing and response characteristics.

⇒ It is commonly used in embedded Systems, automotive control systems, industrial automation, robotics, and other real-time applications.

⇒ RTOS guarantees that tasks or processes will complete within specific time constraints, making it suitable for time-critical applications.

⇒ Examples of RTOS include FreeRTOS, VxWorks, and QNX.

GPOS:-

→GPOS, also known as desktop or server operating systems, is designed for general computing tasks and is not optimized for real-time or deterministic behaviour.

→ GPOS provides multitasking, memory management, file systems, and various application services for a wide range of software.

⇒ Examples of GPOS include windows, macOS, Linux, and Android.

→ GPOS may have non-deterministic response times, making them unsuitable for applications with strict timing requirements.

In summary, the primary difference between RTOS and GPOS is their suitability for real-time and general purpose computing respectively. RTOS is used when precise timing and determinism ~~action~~ are critical, while GPOS is used for everyday computing tasks.

5) Compilers and Interpreters:-

Compilers and interpreters are both tools used in the field of computer programming to convert human-readable code into machine executable code, but they work in different ways.

i) Compiler:-

⇒ A compiler translates the entire source code of a program all at once into machine code or an intermediate representation

⇒ It checks the code for errors and generates a list of all errors found before producing an executable file

→ Once compiled, the program can be executed repeatedly without recompilation, which can make it faster in some cases.

→ Examples of compiled languages include C, C++, and Rust.

Interpreter:-

→ An interpreter translates source code line by line and executes it immediately without generating a separate executable file.

→ It stops at the first error encountered, which allows for easier debugging of small sections of code.

⇒ Interpreted languages tend to be more flexible but Can be slower as they need to translate and execute code simultaneously.

⇒ Examples of interpreted languages include python, javascript, and Ruby.