1: What is Hashing in C#?

Hashing is a process of converting data (like a string, object, or file) into a fixed-size **hash code** — usually a number or byte array — using a **hash function**.

It's used for:

- Fast **data lookup** (e.g., in dictionaries)
- **Comparing objects** (e.g., passwords)
- **Storing data securely** (e.g., password hashing)

➢ **Key Concepts:**

Term	Description
Hash Function	**A method that takes input and returns a fixed-size hash code (int or bytes).**
Hash Code	**The output (number or byte array) from the hash function.**
Deterministic	**The same input always gives the same hash code.**
Collision	**When two different inputs produce the same hash (rare, but possible).**

➢ **Real-World Uses:**

Scenario	Hashing Role
Password Storage	**Store hash instead of plain password**
File Integrity Check	**Compare hash before/after transfer**
Hash Tables (Dictionaries)	**Quickly find key-value pairs**
Caching / Deduplication	**Identify data using its hash code**

2: **What is .rdata?**

In compiled applications (like those compiled with C++ or even when analyzing C#'s native output), .rdata stands for **Read-Only Data Section**. It contains:

- **String literals**
- **Constants**
- **VTables** (in C++)
- Debug info (sometimes)
- Metadata used at runtime (like type names, attributes in C#)

In C#, while we don't work with .rdata directly, tools like **ILSpy** or **dnSpy** let you reverse-engineer compiled .NET assemblies, and you may spot parts of this.

3: **Switch Evolution in C#**

C#’s switch started simple and became way smarter over time:

- **C# 1–5:**
  Classic switch — only works with constants like numbers, chars, and later strings.
- **C# 7:**
  Introduced **pattern matching**. You can check types and use when conditions.
- **C# 8:**
  Added **switch expressions** for cleaner, one-line results.
- **C# 9+**:
  Powerful **property & relational patterns**, and/or logic, and nested patterns.

➢ **Why C# is strong here:**
Modern switch is shorter, expressive, supports pattern matching, and replaces many if-else chains cleanly.

4: Where C# Has the Upper Hand in the C-Family (C, C++, Java)

C# started as a **modern member of the C family** and improved on many pain points of C/C++/Java. Here's a clean summary:

- **Memory Management**
  - **C#:** Automatic **Garbage Collection**, no manual free() or delete.
  - **C/C++:** Manual memory handling → memory leaks & dangling pointers.
    Upper hand: Safer and easier for developers.

- **Rich switch and Pattern Matching**
  - **C#:** Modern switch → expressions, type patterns, property patterns.
  - **C/C++/Java:** Classic switch → limited to constants or enums (Java's switch improved only recently).
    Upper hand: C# reduces verbose if-else chains and adds expressive pattern matching.

- **Managed Code & CLR**
  - **C#:** Runs on the **.NET CLR** → type safety, JIT optimization, and security.
  - **C++:** Native → faster raw performance, but less safe.
    Upper hand: Safety + productivity for business apps and backend.

- **Language Features**
  - **Properties, Events, Delegates** → Built-in OOP event system.
  - **LINQ** → Query collections like SQL, not in C++ or old Java.
  - **Async/Await** → Cleaner async programming than callbacks or threads in C++.

- **Cross-Platform & Rapid Development**
  - With **.NET 5+ / .NET Core**, C# runs on Windows, Linux, macOS.
  - Easy **GUI, Web, and API** development compared to raw C++ setup.

❖ **Quick Takeaway**

C# trades **a bit of raw speed** for:

- Safety 🛡
- Cleaner syntax ✨
- Productivity 🚀
- Modern features like **pattern matching, LINQ, async/await**

It shines in **enterprise apps, game dev (Unity), and backend APIs** while still feeling familiar to anyone from the **C family**.