**C# Fundamentals**

**1. \*\*What is C# and what are its key features?\*\***

- C# is a modern, object-oriented programming language developed by Microsoft for .NET. Key features include:

- Strong typing

- Garbage collection

- LINQ (Language Integrated Query)

- Asynchronous programming

**- \*\*Example\*\*:** C# uses CLR (Common Language Runtime) to manage memory automatically.

**2. \*\*Explain the basic structure of a C# program.\*\***

- A basic C# program includes:

- \*\*Namespace\*\*: Groups related classes.

- \*\*Class\*\*: Holds methods and variables.

- \*\*Main Method\*\*: Entry point.

**- \*\*Example\*\*:**

```csharp

namespace HelloWorldApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World");

}

}

}

```

**3. \*\*What are the different types of data types available in C#?\*\***

- \*\*Value Types\*\*: int, float, bool, struct.

- \*\*Reference Types\*\*: string, array, class, delegate.

**4. \*\*What is the difference between value types and reference types?\*\***

- \*\*Value types\*\* store data directly, while \*\*reference types\*\* store a reference to the data.

**- \*\*Example\*\*:** `int` is a value type; `string` is a reference type.

**5. \*\*What are nullable types in C#?\*\***

- Nullable types allow value types to represent null values.

**- \*\*Example\*\*: `**int? x = null;`

**6. \*\*Can you describe what namespaces are and how they are used in C#?\*\***

- Namespaces group related classes, preventing naming conflicts.

**- \*\*Example\*\*:** `System` namespace includes `Console`.

**7. \*\*Explain the concept of boxing and unboxing in C#.\*\***

- \*\*Boxing\*\*: Converting a value type to a reference type.

- \*\*Unboxing\*\*: Converting a reference type back to a value type.

**- \*\*Example\*\*:**

```csharp

int x = 10;

object obj = x; // Boxing

int y = (int)obj; // Unboxing

```

**8. \*\*What is Type Casting and what are its types in C#?\*\***

- Type casting converts data from one type to another.

- \*\*Types\*\*: Implicit (automatic) and Explicit (using casting).

**- \*\*Example\*\*:** `(int)3.5` is explicit casting.

**9. \*\*What are operators in C# and can you provide examples?\*\***

- Operators perform actions on variables.

**- \*\*Example\*\*:** Arithmetic `+`, Relational `==`, Logical `&&`.

**10. \*\*What is the difference between == operator and Equals() method?\*\***

- `==` checks value equality, while `Equals()` checks content equality.

**- \*\*Example\*\*:** For strings, `Equals()` checks content, `==` can also check reference.

**11. \*\*What is the purpose of the var keyword in C#?\*\***

- `var` infers the type at compile time.

**- \*\*Example\*\*:** `var x = 10;` infers `x` as `int`.

**12. \*\*What are the differences between const and readonly keywords?\*\***

- `const` is compile-time constant; `readonly` is runtime constant.

**- \*\*Example\*\*:**

```csharp

const int x = 10;

readonly int y = DateTime.Now.Year;

```

**13. \*\*How does checked and unchecked context affect arithmetic operations?\*\***

- \*\*Checked\*\*: Throws exception on overflow.

- \*\*Unchecked\*\*: Ignores overflow.

**- \*\*Example\*\*:**

```csharp

int x = checked(2147483647 + 1); // Exception

```

**14. \*\*What are the different ways to handle errors in C#?\*\***

- Using `try-catch`, `finally`, and custom exception classes.

**- \*\*Example\*\*:**

```csharp

try { int x = 1 / 0; }

catch (Exception ex) { Console.WriteLine(ex.Message); }

```

**15. \*\*Explain the role of the garbage collector in .NET.\*\***

- Garbage collector frees memory by deleting unreferenced objects, improving memory management.

**### Object-Oriented Programming in C#**

**16. \*\*Define Object-Oriented Programming and its principles.\*\***

- OOP is a programming paradigm based on objects and data, using \*\*Encapsulation\*\*, \*\*Inheritance\*\*, \*\*Polymorphism\*\*, and \*\*Abstraction\*\*.

**17. \*\*What is a class and how is it different from a struct?\*\***

- A class is a reference type supporting inheritance; a struct is a value type without inheritance.

**18. \*\*Explain the concept of inheritance and its use in C#.\*\***

- Inheritance allows a class to inherit properties/methods from another.

**- \*\*Example\*\*:**

```csharp

class Animal { }

class Dog : Animal { }

```

**19. \*\*What is polymorphism, and can you give a C# example?\*\***

- Polymorphism allows methods to perform differently based on the object.

**- \*\*Example\*\*:**

```csharp

Animal animal = new Dog();

animal.Speak(); // Dog's Speak() called

```

**20. \*\*What is encapsulation and how is it implemented in C#?\*\***

- Encapsulation restricts access to data, implemented using `private` and `public` access modifiers.

**- \*\*Example\*\*:**

```csharp

class Account { private int balance; public void Deposit(int amount) { balance += amount; } }

```

**21. \*\*What are abstract classes and interfaces, and when do you use each?\*\***

- \*\*Abstract class\*\*: Contains common behavior and some implementations.

- \*\*Interface\*\*: Defines a contract without implementations.

**- \*\*Example\*\*:**

```csharp

interface IMovable { void Move(); }

abstract class Animal { public abstract void Speak(); }

```

**22. \*\*Can you explain what a virtual method is in C#?\*\***

- A virtual method can be overridden in derived classes.

**- \*\*Example\*\*:**

```csharp

class Animal { public virtual void Speak() => Console.WriteLine("Animal speaks"); }

```

**23. \*\*What is method overloading and method overriding?\*\***

- \*\*Overloading\*\*: Same method name, different parameters.

- \*\*Overriding\*\*: Derived class changes the base class method behavior.

**- \*\*Example\*\*:**

```csharp

void Print(int i) { }

void Print(double d) { } // Overloading

```

**24. \*\*Can you describe the base keyword?\*\***

- \*\*Base\*\* is used to access base class members in a derived class.

**- \*\*Example\*\*:**

```csharp

class Dog : Animal { public Dog() : base() { } }

```

**25. \*\*What is an access modifier and what are the different types of access modifiers?\*\***

- Access modifiers define the visibility of classes and members:

- `public`, `private`, `protected`, `internal`, `protected internal`, `private protected`.

Here are concise, detailed answers with examples for the advanced C# concepts and collections/data structures interview questions:

---

**### C# Advanced Concepts**

**26. \*\*What are indexers in C#?\*\***

- Indexers allow instances of a class to be indexed like arrays.

**- \*\*Example\*\*:**

```csharp

class SampleCollection

{

private string[] arr = new string[10];

public string this[int index]

{

get => arr[index];

set => arr[index] = value;

}

}

```

**27. \*\*Explain the concept of delegates in C#.\*\***

- A delegate is a type-safe function pointer, allowing methods to be passed as parameters.

**- \*\*Example\*\*:**

```csharp

public delegate void PrintDelegate(string message);

public void PrintMessage(string msg) => Console.WriteLine(msg);

PrintDelegate del = PrintMessage;

del("Hello");

```

**28. \*\*What are events and how are they different from delegates?\*\***

- Events are a way for a class to notify other classes when something occurs. They use delegates but add constraints for safer publishing.

**- \*\*Example\*\*:**

```csharp

public event PrintDelegate OnPrint;

```

**29. \*\*What are Lambda expressions and where would you use them?\*\***

- Lambda expressions are concise methods used in LINQ queries and event handling.

**- \*\*Example**\*\*: `nums.Where(x => x > 5);`

**30. \*\*Can you explain what extension methods are and how to use them?\*\***

- Extension methods add functionality to existing types without modifying them.

**- \*\*Example\*\*:**

```csharp

public static class StringExtensions { public static bool IsEmpty(this string s) => s.Length == 0; }

```

**31. \*\*What are generics and how do they provide type safety?\*\***

- Generics allow code to operate on different types while enforcing type safety.

**- \*\*Example\*\*:**

```csharp

List<int> list = new List<int>();

```

**32. \*\*Define LINQ and mention its advantages.\*\***

- LINQ (Language Integrated Query) enables querying collections in a readable way, supporting compile-time checking and IntelliSense.

**- \*\*Example\*\*:** `var result = numbers.Where(n => n > 5);`

**33. \*\*What is the difference between IEnumerable and IQueryable?\*\***

- `IEnumerable` executes in-memory; `IQueryable` allows remote, dynamic query processing (like database queries).

**34. \*\*What are async and await keywords and how do they work?\*\***

- `async` and `await` enable asynchronous programming, allowing tasks to run without blocking.

**- \*\*Example\*\*:**

```csharp

async Task<int> GetDataAsync() { return await Task.FromResult(10); }

```

**35. \*\*What is the purpose of the using statement?\*\***

- `using` ensures resources are disposed of correctly after use.

**- \*\*Example\*\*:**

```csharp

using (var sw = new StreamWriter("file.txt")) { sw.WriteLine("Hello"); }

```

---

**### C# Collections and Data Structures**

**36. \*\*What are collections in C#?\*\***

- Collections are data structures used to manage groups of objects, like lists, dictionaries, and queues.

**37. \*\*What is the difference between arrays and collections?\*\***

- Arrays have fixed sizes and types, while collections (e.g., List) are dynamic and offer more methods.

**38. \*\*Explain the different types of collections in .NET.\*\***

- \*\*List\*\*: Dynamic array.

- \*\*Dictionary\*\*: Key-value pairs.

- \*\*Queue\*\*: FIFO.

- \*\*Stack\*\*: LIFO.

**39. \*\*What is the difference between List and LinkedList?\*\***

- \*\*List\*\*: Array-backed, fast random access.

- \*\*LinkedList\*\*: Node-based, efficient insertions/deletions.

**40. \*\*Can you discuss the IDictionary interface and its implementation?\*\***

- `IDictionary` defines a key-value pair collection. `Dictionary` and `SortedList` are common implementations.

**41. \*\*What are HashTable and Dictionary and how do they differ?\*\***

- \*\*HashTable\*\*: Non-generic, thread-safe.

- \*\*Dictionary\*\*: Generic, better performance in .NET.

**42. \*\*How does a C# HashSet work and what are its benefits?\*\***

- `HashSet` stores unique values only, useful for eliminating duplicates.

**43. \*\*What are Enumerable and Queryable collections?\*\***

- `Enumerable` supports LINQ in-memory; `Queryable` is for external data sources.

**44. \*\*When would you use a Queue vs a Stack?\*\***

- \*\*Queue\*\*: FIFO, like task scheduling.

- \*\*Stack\*\*: LIFO, like undo operations.

**45. \*\*How do you sort elements in a collection?\*\***

- Use `Sort()` method in lists or LINQ `OrderBy()`.

**- \*\*Example\*\*:**

```csharp

list.Sort();

```

---

### C# Exception Handling

**46. \*\*What is exception handling and why is it necessary?\*\***

- Exception handling manages runtime errors to ensure application stability.

**47. \*\*What are the common exception types in C#?\*\***

- `NullReferenceException`, `IndexOutOfRangeException`, `InvalidOperationException`, etc.

**48. \*\*How do you create custom exceptions in C#?\*\***

- Create a class inheriting from `Exception`.

**- \*\*Example\*\*:**

```csharp

public class CustomException : Exception { }

```

**49. \*\*What is the use of the finally block?\*\***

- `finally` runs code regardless of whether an exception occurs, typically for cleanup.

**50. \*\*Can you explain exception filters introduced in C# 6?\*\***

- Exception filters let you catch specific exceptions based on conditions.

**- \*\*Example\*\*:**

```csharp

catch (Exception ex) when (ex.Message.Contains("specific"))

```

**### C# Asynchronous Programming**

**51. \*\*What is the Task Parallel Library (TPL)?\*\***

- TPL simplifies parallel and asynchronous tasks, providing constructs like `Task` and `Parallel` for concurrency.

**- \*\*Example\*\*:** `Task.Run(() => Console.WriteLine("Hello TPL"));`

**52. \*\*Explain the difference between synchronous and asynchronous operations.\*\***

- \*\*Synchronous\*\*: Blocks the thread until a task completes.

- \*\*Asynchronous\*\*: Frees the thread, allowing other tasks to run concurrently.

**53. \*\*How do you cancel an asynchronous operation?\*\***

- Use a `CancellationToken` with tasks to signal cancellation.

**- \*\*Example\*\*:**

```csharp

var cts = new CancellationTokenSource();

Task.Run(() => { /\* check cts.Token \*/ }, cts.Token);

cts.Cancel();

```

**54. \*\*What is the difference between Task and Thread?\*\***

- \*\*Task\*\*: Managed by TPL, optimized for parallelism.

- \*\*Thread\*\*: OS-level construct with more overhead.

**55. \*\*Discuss the use of the Parallel class in C#.\*\***

- `Parallel` executes loops concurrently for data processing.

**- \*\*Example\*\*:**

```csharp

Parallel.For(0, 10, i => Console.WriteLine(i));

```

---

**### C# File I/O and Serialization**

56. \*\*How do you read from and write to a text file in C#?\*\*

- Use `File.ReadAllText` or `File.WriteAllText`.

**- \*\*Example\*\*:**

```csharp

File.WriteAllText("file.txt", "Hello");

var text = File.ReadAllText("file.txt");

```

**57. \*\*What are the file handling classes in C#?\*\***

- `File`, `FileInfo`, `StreamReader`, `StreamWriter`, `FileStream`.

**58. \*\*Explain serialization and deserialization in the context of C#.\*\***

- \*\*Serialization\*\*: Converts an object to a byte stream.

- \*\*Deserialization\*\*: Converts byte stream back to an object.

**- \*\*Example\*\*:**

```csharp

var json = JsonSerializer.Serialize(myObject);

var obj = JsonSerializer.Deserialize<MyClass>(json);

```

**59. \*\*What is the difference between XML Serialization and JSON Serialization?\*\***

- \*\*XML Serialization\*\*: XML format, supports only public members.

- \*\*JSON Serialization\*\*: JSON format, faster, supports complex types.

**60. \*\*How do you use streams in C#?\*\***

- Use streams (`FileStream`, `MemoryStream`) for byte-level I/O operations.

**- \*\*Example\*\*:**

```csharp

using var fs = new FileStream("file.txt", FileMode.Open);

```

---

**### C# Attributes and Reflection**

**61. \*\*What are attributes in C#?\*\***

- Attributes provide metadata about program elements, like `[Serializable]`.

**62. \*\*How do you define a custom attribute?\*\***

- Create a class inheriting `Attribute`.

**- \*\*Example\*\*:**

```csharp

[AttributeUsage(AttributeTargets.Class)]

public class MyCustomAttribute : Attribute { }

```

**63. \*\*What is reflection and why is it useful?\*\***

- Reflection inspects assembly metadata at runtime, useful for dynamic behavior.

**64. \*\*Explain how to use reflection to inspect an assembly's metadata.\*\***

- Use `Assembly` class to load and inspect types and members.

**- \*\*Example\*\*:**

```csharp

Assembly assembly = Assembly.Load("MyAssembly");

```

**65. \*\*How do you use reflection to create an instance of a class at runtime?\*\***

- Use `Activator.CreateInstance`.

**- \*\*Example\*\*:**

```csharp

var instance = Activator.CreateInstance(typeof(MyClass));

```

---

**### C# Memory Management**

**66. \*\*Describe the stack and heap in .NET's memory management.\*\***

- \*\*Stack\*\*: Stores value types, fixed size.

- \*\*Heap\*\*: Stores reference types, dynamic allocation.

**67. \*\*What are the finalizers in C#?\*\***

- Finalizers (`~ClassName`) clean up unmanaged resources when the GC collects an object.

**68. \*\*How do you force a garbage collection?\*\***

- Use `GC.Collect()`, though it’s generally discouraged due to performance impact.

**69. \*\*Explain the IDisposable interface and the Dispose pattern.\*\***

- `IDisposable` releases unmanaged resources using `Dispose()`.

**- \*\*Example\*\*:**

```csharp

public void Dispose() { /\* cleanup code \*/ }

```

**70. \*\*What is a memory leak in .NET and how can it be prevented?\*\***

- Memory leaks occur when objects aren’t properly released. Avoid holding references longer than needed.

---

**### C# Debugging and Diagnostic**

**71. \*\*How do you debug a C# application?\*\***

- Use Visual Studio debugger, breakpoints, and `Debug.WriteLine`.

**72. \*\*What are breakpoints and how are they used?\*\***

- Breakpoints pause code at specific lines for inspection.

**73. \*\*Explain the use of the Debug and Trace classes.\*\***

- `Debug` (development) and `Trace` (production) are for diagnostic logging.

**74. \*\*Discuss the techniques to analyze a memory dump.\*\***

- Use tools like WinDbg or Visual Studio to analyze memory allocation and object lifetimes.

**75. \*\*How can you profile a C# application to identify performance bottlenecks?\*\***

- Use profilers like Visual Studio’s Performance Profiler to analyze CPU and memory usage.

Here are brief, detailed answers with examples for C# concurrency, parallelism, unit testing, TDD, best practices, design patterns, interoperability, and future evolution:

---

**### C# Concurrency and Parallelism**

**76. \*\*What is a deadlock and how can it be prevented?\*\***

- Deadlock occurs when threads wait indefinitely for resources held by each other. Prevention includes using timeouts or ordered resource acquisition.

**77. \*\*Discuss the reader-writer lock pattern in C#.\*\***

- Reader-writer locks allow multiple readers but only one writer. Use `ReaderWriterLockSlim` for high concurrency scenarios.

**78. \*\*Explain how the lock keyword ensures thread safety.\*\***

- `lock` prevents simultaneous access to shared resources by serializing access.

**- \*\*Example\*\*:**

```csharp

lock (syncObject) { /\* thread-safe code \*/ }

```

**79. \*\*What are Mutexes, Semaphores, and Monitors?\*\***

- \*\*Mutex\*\*: System-wide locking.

- \*\*Semaphore\*\*: Limits threads accessing resources.

- \*\*Monitor\*\*: Synchronization for .NET objects (e.g., `Monitor.Enter`).

**80. \*\*How do you achieve parallelism using PLINQ?\*\***

- Use `AsParallel()` with LINQ for parallel data processing.

**- \*\*Example\*\*:**

```csharp

var result = numbers.AsParallel().Where(n => n % 2 == 0).ToList();

```

---

**### C# Unit Testing and Test Driven Development (TDD)**

**81. \*\*What is unit testing and what frameworks do you use for it in C#?\*\***

- Unit testing verifies small code units. Common frameworks include MSTest, NUnit, and xUnit.

**82. \*\*Explain the concept of Test-Driven Development (TDD).\*\***

- TDD involves writing tests before code, aiming for iterative, testable design.

**83. \*\*How do you mock objects in C# unit tests?\*\***

- Use `Moq` or `NSubstitute` to create mock dependencies.

**- \*\*Example\*\*:**

```csharp

var mock = new Mock<IMyService>();

```

**84. \*\*What are the common attributes used in a test method?\*\***

- `[TestMethod]`, `[Fact]`, `[Theory]` (for parameterized tests), `[Setup]` for initialization.

**85. \*\*How do you test asynchronous code in C#?\*\***

- Use async methods and `await` in test frameworks.

**- \*\*Example\*\*:**

```csharp

public async Task MyAsyncTest() { await myAsyncMethod(); }

```

---

**### C# Best Practices and Design Patterns**

**86. \*\*Why are SOLID principles important in C#?\*\***

- SOLID principles enhance code maintainability and flexibility, preventing dependency issues.

**87. \*\*Can you describe some common design patterns and their applications in C#?\*\***

- \*\*Singleton\*\*: Ensures one instance of a class.

- \*\*Factory\*\*: Creates objects based on type.

- \*\*Observer\*\*: Manages dependency changes (e.g., event handling).

**88. \*\*How do you ensure your C# code is maintainable and readable?\*\***

- Follow naming conventions, refactor regularly, use design patterns.

**89. \*\*What strategies do you use for error handling and exception management?\*\***

- Use `try-catch`, specific exceptions, and logging frameworks like Serilog.

**90. \*\*Discuss the concept of dependency injection and how it's used in C#.\*\***

- DI manages dependencies by injecting them, often using frameworks like Autofac or built-in .NET Core DI.

**91. \*\*What are the new features introduced in the latest version of C#?\*\***

- Updates include improved pattern matching, records, and performance enhancements.

**92. \*\*How has pattern matching evolved in recent C# versions?\*\***

- Adds more versatile matching with `switch` expressions, tuples, and nested patterns.

**93. \*\*Explain how C# 8 nullable reference types work.\*\***

- `Nullable` annotations identify nullable references, reducing `NullReferenceException` risks.

**94. \*\*What is the switch expression and how does it differ from the switch statement?\*\***

- `switch` expression returns values directly and uses concise syntax.

**95. \*\*How do you take advantage of tuples in C#?\*\***

- Tuples store multiple values without custom classes.

**- \*\*Example\*\*:**

**```csharp**

**var tuple = (1, "item");**

**```**

---

**### C# Interoperability**

**96. \*\*How can you call unmanaged code using C#?\*\***

- Use `DllImport` for P/Invoke to call unmanaged DLL functions.

**97. \*\*What is the role of P/Invoke in C#?\*\***

- P/Invoke invokes unmanaged code from managed C# code using `DllImport`.

**98. \*\*How do you interface with COM objects in C#?\*\***

- Use `dynamic` and `Interop` libraries to interact with COM components.

**99. \*\*Discuss C# and .NET Core inter-platform capabilities.\*\***

- .NET Core enables cross-platform development for Windows, macOS, and Linux.

---

**### C# and .NET Framework/Core Future**

**100. \*\*How is C# evolving with .NET 5 and beyond?\*\***

- C# is focused on performance, pattern matching, and cross-platform capabilities with improvements in language features and .NET unification.

---