1. \*\*.NET Framework\*\* - это платформа разработки приложений, разработанная Microsoft, которая предоставляет среду выполнения и библиотеки классов для разработки и запуска разнообразных приложений. Он состоит из следующих компонентов:

- \*\*Common Language Runtime (CLR)\*\*: Основная часть .NET Framework, отвечает за выполнение кода и управление памятью.

- \*\*Framework Class Library (FCL)\*\* или \*\*Base Class Library (BCL)\*\*: Это набор библиотек, предоставляющих множество готовых классов и функциональность для разработки приложений.

- \*\*Common Intermediate Language (CIL)\*\*: Промежуточный байт-код, в который компилируется исходный код .NET.

- \*\*Common Language Infrastructure (CLI)\*\*: Стандарт, описывающий архитектуру и правила для создания, компиляции и выполнения .NET-кода.

2. \*\*CLR (Common Language Runtime)\*\* - это часть .NET Framework, которая управляет выполнением кода, обеспечивает сборку мусора и другие важные функции. \*\*FCL/BCL (Framework Class Library/Base Class Library)\*\* - это набор библиотек классов для разработки .NET-приложений. \*\*CLI (Common Language Infrastructure)\*\* - это стандарт, описывающий архитектуру .NET-платформы. \*\*IL (Intermediate Language)\*\* - это байт-код, в который компилируется исходный код для выполнения в CLR.

3. \*\*JIT-компилятор (Just-In-Time Compiler)\*\* - это компонент CLR, который компилирует CIL-код (байт-код) в нативный машинный код во время выполнения программы. Это позволяет повысить производительность, так как машинный код выполняется быстрее, чем интерпретация CIL-кода.

4. \*\*CTS (Common Type System)\*\* - это стандарт, определяющий типы данных и их поведение в рамках .NET Framework. Он обеспечивает согласованное использование типов в разных .NET-языках.

5. \*\*System.Object\*\* определяет базовое поведение для всех типов в .NET. Он имеет методы, такие как Equals, GetHashCode, GetType, и ToString, которые могут быть переопределены в производных типах.

6. \*\*mscorlib.dll\*\* - это библиотека, содержащая базовые классы и типы .NET Framework. В ней содержатся классы, такие как System.Object, System.String и многие другие, необходимые для работы .NET-приложений.

7. \*\*Сборка\*\* в .NET - это единица развертывания и управления версиями. Она может содержать один или несколько файлов, включая исполняемый код, метаданные и ресурсы. Сборка состоит из одного или нескольких модулей.

8. Существуют два основных вида сборок:

- \*\*Сборки приложений (Application Assemblies)\*\*: Это сборки, содержащие исполняемый код конкретного приложения.

- \*\*Сборки библиотек (Library Assemblies)\*\*: Это сборки, содержащие библиотечные классы и ресурсы, которые могут быть использованы другими приложениями.

9. \*\*Assembly Manifest\*\* - это часть сборки, которая содержит метаданные о сборке, включая версию, список файлов, зависимости и другую информацию, необходимую для управления сборкой.

10. \*\*GAC (Global Assembly Cache)\*\* - это централизованное хранилище сборок в .NET. Он используется для разделения и управления общими сборками, доступными для разных приложений на компьютере.

11. \*\*Managed code\*\* - это код, который выполняется в среде с управлением CLR и подвергается автоматическому управлению памятью. \*\*Unmanaged code\*\* - это код, который выполняется вне среды с управлением и обычно не подвергается автоматическому управлению памятью.

12. Метод \*\*Main\*\* является точкой входа для приложения .NET. Он вызывается при запуске приложения и используется для начальной инициализации и выполнения программы.

13. Директива \*\*using\*\* в C# используется для объявления и использования пространств имен. Она позволяет упростить доступ к типам и членам этих пространств имен. Примеры:

```csharp

using System; // Импорт пространства имен System

using MyNamespace; // Импорт пользовательского пространства имен

```

14. Сборки и пространства имен связаны так, что пространства имен могут содержать типы, а типы могут быть частью сборок. Сборки предоставляют логическую группировку типов и метаданные о типах.

15. \*\*Примитивные типы данных\*\* - это базовые типы данных, предоставляемые языком C#. Они включают:

- Целочисленные типы: int, long, short, byte, sbyte, uint, ulong, ushort.

- Типы с плавающей точкой: float, double.

- Логический тип: bool.

- Символьный тип

: char.

- Типы для работы с текстом: string.

16. \*\*Ссылочные типы\*\* - это типы данных, которые хранят ссылку на объект в памяти. Они включают классы, интерфейсы, делегаты и массивы.

17. \*\*Типы-значения\*\* - это типы данных, которые хранят свое значение напрямую, а не ссылку на объект. Они включают числовые типы, структуры и перечисления (enum).

18. Основное отличие между ссылочными и значимыми типами данных заключается в способе хранения и передачи данных. Ссылочные типы передаются по ссылке, а значимые типы передаются по значению. Это означает, что изменения значимых типов в одном месте не затрагивают их в другом месте, в то время как изменения ссылочных типов могут влиять на все ссылки на объект.

19. \*\*Упаковка (boxing)\*\* - это процесс преобразования значимого типа данных в ссылочный тип (object). \*\*Распаковка (unboxing)\*\* - это процесс обратного преобразования объекта (object) обратно в значимый тип данных. Упаковка может вызвать накладные расходы по производительности.

20. Разница между `int` и `System.Int32`, а также `double` и `System.Double`, заключается в том, что `int` и `double` - это синонимы имен типов, предоставляемых C#, в то время как `System.Int32` и `System.Double` - это полные имена типов из пространства имен System.

21. Тип `dynamic` используется для определения переменных, которые могут хранить значения различных типов данных во время выполнения. Это позволяет обойти статическую типизацию в C# и выполнять динамическое разрешение типов.

22. Основное отличие между `var` и `dynamic` заключается в том, что `var` используется для объявления переменных с неявной типизацией на этапе компиляции, в то время как `dynamic` используется для переменных с динамической типизацией на этапе выполнения.

23. \*\*Неявно типизированная переменная\*\* - это переменная, объявленная с использованием ключевого слова `var`, которое позволяет компилятору определить тип переменной на основе ее инициализации.

24. \*\*Nullable тип\*\* (типы-значения с `?`) используется для хранения значений, которые могут быть равны `null`. Он предоставляет способ обрабатывать отсутствующие значения для типов, которые обычно не допускают `null`, такие как `int` или `DateTime`.

25. Строковый литерал объявляется с использованием двойных кавычек, например: `"Hello, World!"`. Операции над строками включают конкатенацию (соединение) строк, поиск подстроки, разделение строки и другие.

26. Строки можно задавать и инициализировать следующими способами:

- С использованием строковых литералов: `string str = "Hello, World!";`

- С использованием конструктора: `string str = new string('A', 5);`

27. Некоторые методы у типа `String` включают `Length` (длина строки), `Concat` (конкатенация), `Substring` (выделение подстроки), `IndexOf` (поиск подстроки), `Replace` (замена подстроки) и многие другие.

28. \*\*Пустая строка\*\* - это строка, которая содержит ноль символов, но она всегда является объектом строки. \*\*Null строка\*\* - это ссылка на строку, которая не указывает на какой-либо объект строки и имеет значение `null`.

29. Сравнение строк можно выполнить с использованием операторов `==` (сравнение по значению) и `Equals` (сравнение по содержимому), а также с использованием методов `Compare` и `CompareTo` для определения относительного порядка строк.

30. \*\*String\*\* представляет неизменяемую (immutable) строку, что означает, что каждая операция изменения строки создает новый экземпляр строки, в то время как \*\*StringBuilder\*\* представляет изменяемую строку, позволяя эффективно выполнять множество операций изменения.

31. \*\*Методы Convert\*\* используются для преобразования значений между различными типами данных, например, из строки в число или из числа в строку. Пример:

```csharp

int number = Convert.ToInt32("123");

```

32. Консольный ввод/вывод можно выполнить с использованием класса `Console`. Например, `Console.WriteLine("Hello, World!");` выводит текст на консоль, а `Console.ReadLine()` читает строку из консоли.

33. Примеры определения и инициализации одномерного и двумерного массивов:

- Одномерный массив: `int[] numbers = new int[5];`

- Двумерный массив: `int[,] matrix = new int[3, 3];`

34. \*\*Ступенчатый массив\*\* - это массив массивов, где каждый элемент может быть другим массивом. Он создается с помощью вложенных массивов разной длины.

35. В цикле `foreach` можно использовать объекты, реализующие интер

фейс `IEnumerable`, такие как массивы, списки и другие коллекции. Пример:

```csharp

foreach (var item in myArray)

{

// Действия с элементами массива

}

```

36. \*\*Кортеж\*\* - это структура данных, которая позволяет объединять несколько значений разных типов в одну переменную. Он используется для возврата нескольких значений из метода или для удобного группирования данных.

37. \*\*Локальная функция\*\* - это функция, объявленная внутри другой функции или метода. Она имеет доступ к переменным из внешней функции и может использоваться для улучшения читаемости и организации кода.

38. Код, заключенный в блок `checked`, выполняет проверку на переполнение для арифметических операций. Код, заключенный в блок `unchecked`, отключает проверку на переполнение. По умолчанию, вне блока `checked` или `unchecked`, применяется контекст, определенный в настройках компилятора.

39. Контекст (checked/unchecked) по умолчанию определяется настройками компилятора. Его можно изменить с помощью ключевого слова `checked` или `unchecked`, а также глобальными настройками компилятора.

40. Ключевое слово `fixed` используется для создания фиксированных указателей на управляемые объекты. Это часто используется для взаимодействия с низкоуровневыми операциями, такими как работа с памятью и вызовы нативных функций. Указатели, объявленные с использованием `fixed`, не подвержены сборке мусора и могут управляться напрямую.