1. .NET Framework - это программная платформа от Microsoft, предназначенная для разработки, выполнения и управления приложениями. Он состоит из следующих компонентов:

- CLR (Common Language Runtime): Виртуальная машина, которая управляет выполнением кода на различных языках программирования.

- FCL (Framework Class Library) или BCL (Base Class Library): Библиотека классов, предоставляющая набор классов и методов для решения общих задач, таких как работа с файлами, сетью, графическими интерфейсами и многое другое.

- ASP.NET: Фреймворк для разработки веб-приложений.

- WinForms и WPF: Фреймворки для создания приложений с графическим интерфейсом.

- ADO.NET: Технология для работы с базами данных.

- ASP.NET Core: Следующее поколение фреймворка ASP.NET для кроссплатформенной разработки веб-приложений.

2. CLR (Common Language Runtime) - это виртуальная машина .NET, которая управляет выполнением кода на различных языках программирования. FCL/BCL (Framework/Base Class Library) - это библиотека классов .NET, предоставляющая набор классов и методов для решения общих задач. CLI (Common Language Infrastructure) - это стандарт, описывающий архитектуру .NET, включая спецификации языков, компиляторов и исполнительных сред. IL (Intermediate Language) - это промежуточный язык, на котором компилируется исходный код .NET и который исполняется в CLR.

3. JIT-компилятор (Just-In-Time Compiler) - это компонент CLR, который преобразует код на промежуточном языке (IL) в нативный машинный код во время выполнения программы. Это увеличивает производительность, так как компиляция происходит непосредственно перед выполнением, и код оптимизируется под конкретное оборудование.

4. CTS (Common Type System) - это стандарт, определяющий базовые типы данных и правила взаимодействия типов данных в .NET Framework. Он обеспечивает совместимость между разными языками программирования, поддерживаемыми платформой .NET.

5. System.Object - это базовый класс для всех типов в .NET. Он определяет основные методы, такие как `ToString`, `Equals`, и `GetHashCode`, которые могут использоваться всеми объектами. Этот класс также определяет методы для работы с мониторами, используемыми для синхронизации потоков.

6. mscorlib.dll - это основная сборка (библиотека) .NET Framework, содержащая базовые классы и типы, необходимые для работы приложений .NET.

7. Сборка (Assembly) в .NET - это единица развертывания и управления версиями. Она может включать одно или несколько исполняемых файлов (EXE), библиотек с общим кодом (DLL), метаданные и манифест, описывающий содержание сборки. Сборка .NET может быть развернута, скопирована, загружена и управляется средой CLR.

8. Существуют два основных вида сборок:

- Сборки приложений (Application Assemblies): Сборки, созданные для конкретного приложения. Они могут содержать исполняемый код, библиотеки и другие ресурсы, необходимые для работы приложения.

- Сборки библиотек (Class Library Assemblies): Сборки, созданные для предоставления функциональности, которая может быть использована другими приложениями. Это библиотеки, которые можно повторно использовать.

9. Сборка Manifest (Assembly Manifest) - это часть метаданных сборки, которая содержит информацию о версии сборки, ее зависимостях, культуре и других свойствах. Манифест также содержит список всех файлов, входящих в сборку, и их хеш-суммы для обеспечения целостности.

10. GAC (Global Assembly Cache) - это специальное хранилище сборок, которое позволяет разработчикам устанавливать и обновлять общие библиотеки, доступные для всех приложений на компьютере. Сборки, размещенные в GAC, могут быть

общими и использоваться разными приложениями.

11. Managed code - это код, который выполняется в управляемой среде, такой как CLR. Он обеспечивает автоматическое управление памятью и безопасность выполнения. Unmanaged code - это код, который выполняется вне управляемой среды и может использовать неуправляемые ресурсы и вызывать низкоуровневые API.

12. Метод Main - это точка входа в приложение на языке C#. Он является обязательным инициальным методом, который вызывается при запуске приложения и может принимать аргументы командной строки.

13. Директива `using` в C# используется для импорта пространства имен. Она позволяет использовать классы и другие типы из определенного пространства имен без необходимости полного указания квалифицированного имени. Примеры:

- `using System;` - импорт пространства имен `System`.

- `using System.IO;` - импорт пространства имен `System.IO`.

14. Сборки и пространства имен связаны следующим образом: сборка может содержать одно или несколько пространств имен, а пространство имен может содержать классы, интерфейсы и другие типы.

15. Примитивные типы данных - это базовые типы данных в C#, которые представляют фундаментальные значения. Некоторые из них включают `int`, `double`, `char`, `bool`, `byte`, `short`, `long`, `float`, и другие.

16. Ссылочные типы - это типы данных, которые хранят ссылки на объекты, а не сами объекты. К ним относятся классы, интерфейсы, делегаты, строки и массивы.

17. Типы-значения - это типы данных, которые хранят фактические значения, а не ссылки на объекты. К ним относятся числа, символы, перечисления и структуры.

18. Главное отличие между ссылочными и значимыми типами данных заключается в том, как они хранят и передаются. Ссылочные типы хранятся в куче (heap) и передаются по ссылке, а значимые типы хранятся в стеке (stack) и передаются по значению.

19. Упаковка и распаковка - это процессы преобразования значимых типов (типов-значений) в ссылочные типы и обратно. Упаковка происходит при преобразовании значимого типа в объект (например, `int` в `object`), а распаковка - при извлечении значения из упакованного объекта обратно в значимый тип.

20. Разница между `int` и `System.Int32`, `double` и `System.Double` и т.д. отсутствует. `int` и `double` - это псевдонимы (алиасы) для соответствующих типов `System.Int32` и `System.Double`. Они могут использоваться взаимозаменяемо.

21. Тип `dynamic` позволяет определить переменную, тип которой будет определен во время выполнения программы. Это обеспечивает динамическую типизацию и позволяет выполнять операции без статической проверки типов на этапе компиляции.

22. Разница между `var` и `dynamic` заключается в том, что `var` используется для неявной типизации на этапе компиляции, в то время как `dynamic` обеспечивает динамическую типизацию на этапе выполнения.

23. Неявно типизированная переменная (var) - это переменная, у которой тип определяется компилятором на основе присвоенного значения. Например, `var x = 10;` означает, что `x` будет иметь тип `int`.

24. Nullable тип (типы-значения с вопросительным знаком, например, `int?`) позволяет переменной хранить значение типа-значения или `null`. Это полезно, например, когда в базе данных или вводе данных может отсутствовать значение.

25. Строковый литерал в C# заключается в двойные кавычки, например, `"Hello, world!"`. Операции со строкой включают конкатенацию (`+`), форматирование (`string.Format`), доступ к символам, разделение и многое другое.

26. Строки можно задавать с помощью двойных кавычек (например, `"Hello, world!"`) или с помощью литерала интерполяции (н

апример, `$"Hello, {name}!"`), а также с помощью конструктора строки (`new string("Hello, world!")`).

27. Методы у типа `String` включают `Length` (длина строки), `Substring` (выделение подстроки), `ToLower` (преобразование в нижний регистр), `ToUpper` (преобразование в верхний регистр), `Split` (разделение строки на подстроки) и многие другие.

28. Пустая строка (`""`) - это строка, которая имеет нулевую длину, тогда как null строка (`null`) означает отсутствие ссылки на строку.

29. Сравнение строк можно выполнить с помощью операторов `==` и `!=`, а также с использованием методов `Equals`, `Compare`, `CompareTo` и других. Важно учитывать разницу между сравнением по значению и сравнением по ссылке.

30. `String` и `StringBuilder` оба используются для работы со строками, но `String` является неизменяемым, что означает, что каждая операция изменения создает новую строку, в то время как `StringBuilder` предоставляет изменяемую последовательность символов, что более эффективно для выполнения множественных операций изменения.

31. Явные преобразования с помощью `Convert` используются для преобразования одного типа данных в другой. Например, `int x = Convert.ToInt32("123");` преобразует строку в целое число.

32. Консольный ввод/вывод в C# выполняется с помощью класса `Console`. Для чтения с консоли используется `Console.ReadLine()`, а для вывода - `Console.WriteLine()`.

33. Примеры определения и инициализации массивов:

- Одномерный массив: `int[] numbers = new int[] { 1, 2, 3, 4, 5 };`

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

34. Ступенчатый массив - это массив массивов, где каждый элемент основного массива является массивом. Например:

```csharp

int[][] jaggedArray = new int[3][];

jaggedArray[0] = new int[] { 1, 2, 3 };

jaggedArray[1] = new int[] { 4, 5 };

jaggedArray[2] = new int[] { 6, 7, 8, 9 };

```

35. В цикле `foreach` можно использовать любой тип, который реализует интерфейс `IEnumerable` или `IEnumerable<T>`. Пример:

```csharp

foreach (int number in numbers)

{

Console.WriteLine(number);

}

```

36. Кортеж (Tuple) - это структура данных, позволяющая объединить несколько значений разных типов в один объект. Он может использоваться, например, для возвращения нескольких значений из метода.

37. Локальная функция (Local Function) - это функция, определенная внутри другой функции или метода. Она имеет доступ к переменным и параметрам внешней функции и может быть полезной для организации логики внутри функции.

38. `checked` и `unchecked` - это ключевые слова в C#, используемые для контроля переполнения чисел при выполнении арифметических операций. `checked` активирует проверку переполнения, а `unchecked` отключает ее.

39. По умолчанию контекст выполнения C# активирует проверку переполнения. Это можно изменить, указав ключевое слово `unchecked` перед

блоком кода или использовать настройки компилятора.

40. `fixed` используется для создания указателя на управляемый объект в неуправляемом коде. Он обеспечивает фиксацию объекта в памяти, чтобы избежать его перемещения сборщиком мусора. Это используется в определенных сценариях взаимодействия с неуправляемым кодом, например, при работе с API операционной системы.