1. Что такое .Net Framework и из чего он состоит?

.NET Framework служит средой для поддержки, разработки и выполнения распределенных приложений, которые базируются на компонентах (элементах управления).

Приложения (программы) можно разрабатывать на разных языках программирования, которые поддерживают эту технологию.

.NET Framework обеспечивает:

* совместное использование разных языков программирования;
* безопасность и переносимость программ;
* общую модель программирования на базе платформы Windows.

**.NET Framework** — [программная платформа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0#%D0%BA%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Нужен для запуска приложений, написанных на базе технологии .NET. Последняя предназначена для универсального запуска программных продуктов, написанных на различных языках программирования. Основой платформы является общеязыковая среда исполнения [Common Language Runtime (CLR)](https://ru.wikipedia.org/wiki/Common_Language_Runtime" \o "Common Language Runtime), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Платформа .NET Framework — это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows.

Платформа .NET Framework состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET Framework. Основой платформы .NET Framework является среда CLR.  
2. Что такое CLR, FCL/BCL, CLI, IL?

***CLR****("Common Language Runtime", "общеязыковая исполняющая среда") - это компонент .NET Framework, основной задачей которого является управление интерпретацией и исполнением кода IL. CLR отвечает за изоляцию памяти приложений, проверку типов, безопасность кода, преобразование IL в машинный код.*

***IL****(Intermediate Language) - код, содержащий набор инструкций, не зависящих от платформы. Иными словами, после компиляции исходного кода он преобразуется не в код для какой-то определенной платформы, а в промежуточный код на языке IL.*

**CLR** (Common language runtime) — общеязыковая исполняющая среда. Она обеспечивает интеграцию языков и позволяет объектам благодаря стандартному набору типов и метаданным), созданным на одном языке, быть «равноправными гражданами» кода, написанного на другом.

##### FCL, библиотека классов Framework (библиотека классов Framework)

FCL предоставляет крупномасштабную среду программирования. Это среда, предназначенная для различных приложений.

BCL - это общая среда программирования, называемая библиотекой базовых классов, которую могут использовать разработчики всех языков. Это одна из спецификаций CLI (Common Language Infrastructure), которая в основном включает в себя: выполнение сетевых операций, выполнение операций ввода-вывода, управление безопасностью, текстовые операции, операции с базой данных, операции XML, взаимодействие с журналами событий, отслеживание и некоторые другие.

CLI - разновидность текстового интерфейса между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк, в UNIX-системах возможно применение мыши. Также известен под названиями «консоль» и «терминал»

**IL** (Intermediate Language) — код на специальном языке, напоминающим ассемблер, но написанном для .NET. В него преобразуется код из других языков верхнего уровня (c#, VisualBasic).

CLI, называемый промежуточным языком Microsoft (MSIL) или промежуточным языком (IL). CIL - это промежуточный код, с помощью которого компилятор может компилировать код .NET в среду CLR. Это язык между языками высокого уровня (например, C #) и инструкциями ЦП.

3. Пояснить работу JIT-компилятора?

JIT **читает байт-код из некоторых секторов (редко сразу из всех) и компилирует их в машинный код**. Этим сектором может быть файл, функция или любой фрагмент кода. Единожды скомпилированный код может кэшироваться и в дальнейшем повторно использоваться без перекомпиляции

JIT-компиляция — это способ динамической компиляции, которая запускается после запуска программы и компилирует ее код «на лету». JIT-компиляторы преобразуют код высокоуровневых языков в инструкции, понятные виртуальной машине или процессору.  
4. Что такое CTS (Common Type System)?

**CTS (Common Type System)** — общая система типов в CLR (тип, по-видимому — это аналог класса C#). Это — стандарт, признанный ECMA который описывает определение типов и их поведение. Также определяет правила наследования, виртуальных методов, времени жизни объектов.

CTS (Common Type System) – система типов, которая содержит полное описание всех возможных типов данных и программных конструкций, которые поддерживаются общеязыковой исполнительной средой CLR. Также здесь описывается то, как эти сущности могут взаимодействовать между собою.

Типами могут быть классы, интерфейсы, структуры, перечисления, делегаты.

5. Какие аспекты поведения определяет тип System.Object?()

6. Что находится в mscorlib dll?(все что нашла)

**DLL** - это динамически подключаемая библиотека в Microsoft Windows, содержащая код и данные для общего, многократного и одновременного использования несколькими программами или операционной системой  
7. Что такое «сборка»? Из чего состоит сборка .NET?

**Сборка** является базовой структурной единицей в .NET, на уровне которой проходит контроль версий, развертывание и конфигурация приложения.

Сборки представляют собой базовые элементы развертывания, управления версиями, повторного использования, назначения областей активации и прав доступа для приложений на основе платформы .NET. **Сборка представляет собой коллекцию типов и ресурсов, собранных для совместной работы и образующих логическую функциональную единицу**. Сборки создаются в виде исполняемого файла (EXE) или файла библиотеки динамической компоновки (DLL) и являются стандартными блоками приложений .NET. Они предоставляют сведения для среды CLR, которые нужны для распознавания реализаций типов.

8. Какие виды сборок существуют?

**Сборка** (Assemblies) - самостоятельный, функциональный модуль, "логическая" dll (dll, динамически подгружаемая библиотека - windows стандарт программных модулей). Существует два вида сборок:

* однофайловые сборки;
* многофайловые сборки.

Сборка, которая состоит из одного единого модуля (\*.dll или \*.exe) называется однофайловой. В однофайловых сборках все необходимые CIL-инструкции, метаданные и манифесты размещаются в одном, четко определенном пакете.

Сборка, которая состоит из многих файлов двоичного кода .NET, называется многофайловой. Каждый из этих файлов называется модулем. В многофайловой сборке один из модулей есть главным (primary).

***Сборка****(assembly) - один или несколько файлов, содержащий логический набор функциональности (код и другие данные, связанные с кодом). Бывают* ***статические сборки****, хранящиеся на диске, и* ***динамически****е, которые создаются во время выполнения программы. Сборка - это базовый блок приложения, все ресурсы, относящиеся к ней, доступны или только внутри этого блока, или экспортируются наружу. При выполнении сборка задает область видимости имен и следит за ее соблюдением.*

9. Что такое assembly manifest?

**Манифест** сборки (**assembly** **manifest**) - это внутренняя часть сборки, которая позволяет ей быть самоописанной. **Assembly** **manifest** позволяет идентифицировать сборку, указывает файлы, которые включаются в реализацию сборки, описывает типы и ресурсы, используемые в сборке, указывает зависимости от других сборок, а также набор прав доступа, которые необходимы сборке для корректной работы. *Эта информация используется во время выполнения для разрешения ссылок, проверку корректности версий, проверку целостности загруженных сборок.*  
10.Что такое GAC?

**Global Assembly Cache** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***GAC*** — Глобальный Кэш Сборок) — в инфраструктуре [.NET](https://ru.wikipedia.org/wiki/.NET) — подсистема, хранящая сборки CLI (*CLI assembly*) в централизованном репозитории. Система разработана для борьбы с проблемами разделяемых библиотек, например таких, как [DLL hell](https://ru.wikipedia.org/wiki/DLL_hell).

К сборкам, расположенным в GAC, предъявляется несколько требований. В частности, они должны использовать [strong name](https://ru.wikipedia.org/wiki/Strong_name" \o "Strong name), соблюдать строгую схему указания версий и допускать исполнение нескольких версий кода в рамках единого приложения.

***GAC****— это глобальный кэш сборок. В нем хранятся совместно используемые сборки. Обычно это каталог С:\Windows\Assembly\GAC. Этот каталог имеет определенную структуру, в котором хранятся подкаталоги, имена которых сгенерированы по определенному алгоритму. В GAC можно поместить только сборки со строгими именами. Для того, чтобы поместить сборку в GAC, используют специальный инструмент GACUtil.exe, который знает всю внутреннюю структуру GAC и может генерировать имена подкаталогов надлежащим образом. Регистрировать в GAC сборки необходимо для того, чтобы избежать конфликтов имен сборок. Приведем пример: две компании выпустили сборку и назвали ее одним именем Calculus. Если мы скопируем эту сборку в каталог, в котором уже находится сборка с таким же именем, то мы затрем сборку, которая ранее могла использоваться каким-то приложением. Это приложение с новой сборкой теперь работать не сможет. Решением этой проблемы будет регистрация этих двух сборок в GAC, в котором для каждой будет создан отдельный каталог.*  
11.Чем managed code отличается от unmanaged code

***Управля́емый код*** (*managed code*) — термин, введённый фирмой Microsoft, для обозначения кода программы, исполняемой под «управлением» виртуальной машины .NET — Common Language Runtime или Mono. **При этом машинный код** называется ***неуправля́емым кодом*** (*unmanaged code*).

Слово «*управляемый*» относится к методу обмена информацией между программой и исполняющей средой. Оно означает, что в любой точке исполнения управляющая среда может приостановить исполнение и получить информацию, специфичную для текущего состояния. Необходимая для этого информация представлена в управляемом коде на языке *Intermediate Language* и в связанных с этим кодом метаданных.

**С формальной точки зрения управляемым кодом является любой программный код, исполняемый в среде отладчика**.

12.Как и для чего определен метод Main?

Точкой входа в программу на языке C# является метод Main. Именно с этого метода начинается выполнение программы на C#. И программа на C# должна обязательно иметь метод Main.

Определение метода **Main** обязательно начинается с модификатора static, которое указывает, что метод Main - статический.

Возвращаемым типом метода **Main** обязательно является тип **void**. Кроме того, в качестве параметра он принимает массив строк - string[] args - в реальной программе это те параметры, через которые при запуске программы из консоли мы можем передать ей некоторые значения. Внутри метода располагаются действия, которые выполняет программа.  
13.Варианты использования директивы using( using Directive ) в C#.

Директива using используется в двух случаях:

* разрешает использование типов в пространстве имен, поэтому уточнение использования типа в этом пространстве имен не требуется

using System.Text;

* позволяет создавать псевдонимы пространства имен или типа.Это называется директива using alias.

using Project = PC.MyCompany.Project;

Область директивы using ограничивается файлом, в котором она появляется. **Создание директивы using позволяет использовать типы из пространства имен без указания этого пространства.**

Директива using — импортирует пространство имен, избавляя от необходимости полной квалификации имен стереотипов.

Использовав using System; Нам больше не нужно вводить полный путь, таким образом мы можем вызывать методы быстрее и короче.

Директива using позволяет создавать псевдонимы пространства имен или типа. Это называется директива using alias  
14.Как связаны между собой сборки и пространства имен?

Пространство имен и сборка (файл, в котором реализован тип) не обязательно связаны друг с другом. В частности, различные типы, принадлежащие одному пространству имен, могут быть реализованы в разных сборках. Например, тип System.IO.FileStream реализован в сборке MSCorLib dll, а тип System. IO.FileSystemWatcher — в сборке System dll. На самом деле, сборка System IO dll в .NET Framework даже не поставляется. Одна сборка может содержать типы из разных пространств имен. Например, типы System.Int32 и System.Text.StringBuilder находятся в сборке MSCorLib dll.

Пространство имен предназначено для объединения группы типов, которые связаны между собою с семантической точки зрения. Типы размещаются в сборках (assembly). Под типами понимаются классы, делегаты, интерфейсы, структуры, перечисления.

Мы также можем понять это: пространство имен на самом деле представляет собой группу связанных классов в сборке, например: пространство имен System.Data - это все о типах данных, а пространство имен System.IO - о типах файлового ввода-вывода.  
15.Что такое примитивные типы данных? Перечислите их.

**примитивные** **типы** **данных-это** **набор** **базовых** **типов** **данных, из** **которых** **строятся** **все** **остальные** **типы** **данных**

1. Примитивные типы:
   * Числовые типы (без decimal):
     + Целочисленные со знаком (sbyte, short, int, long)
     + Целочисленные без знака (byte, ushort, uint, ulong)
     + IEEE-представление с плавающей запятой (float, double)
   * Логический тип (bool)
   * Символы Юникода (char)
   * System.IntPtr
   * System.UintPtr

16.Что такое ссылочные типы? Какие типы относятся к ним?

В C# существуют две разновидности типов: ссылочные типы и типы значений. В **переменных ссылочных типов хранятся ссылки на их данные (объекты**), а переменные типа значений содержат свои данные непосредственно. Две переменные ссылочного типа могут ссылаться на один и тот же объект, поэтому операции над одной переменной могут затрагивать объект, на который ссылается другая переменная. Каждая переменная типа значения имеет собственную копию данных, и операции над одной переменной не могут затрагивать другую

**К** **ссылочным** **типам** **в** **C#** **относят:** **классы,** **интерфейсы,** **массивы** **и** **делегаты.**

Ссылочные типы:

* Тип object
* Тип string
* Классы (class)
* Интерфейсы (interface)
* Делегаты (delegate)

17.Какие типы относятся к типам-значениям?

Переменные типа-**значения располагаются в стеке**, что позволяет их быстро создавать и уничтожать. Фактически время жизни такой переменной определяется контекстом, в которой она объявлена. Сама переменная представляется в виде локальной копии. Типы-значения являются классами наследниками от *System.ValueType*, который, в свою очередь, наследуется от *System.Object*. К типам-значениям относятся: **простые типы, типы перечисления, типы структур, типы значений, допускающие *NULL*, типы значений кортежей.**

18. В чем отличие между ссылочными и значимыми типами данных?

В переменных ссылочных типов хранятся ссылки на их данные (объекты), а переменные типа значений содержат свои данные непосредственно  
19.Что такое упаковка и распаковка значимых типов?

Упаковка (boxing) предполагает преобразование объекта значимого типа (например, типа int) к типу object. При упаковке общеязыковая среда CLR обертывает значение в объект типа **System.Object** и сохраняет его в управляемой куче (хипе). Распаковка (unboxing), наоборот, предполагает преобразование объекта типа object к значимому типу. Упаковка и распаковка ведут к снижению производительности, так как системе надо осуществить необходимые преобразования.  
20.В чем заключается разница между int и System.Int32? double и  
System.Double и т.д.?

Разницы никакой, int и Int32 после компиляции в IL код станут одним и тем же.

Единственное различие, вероятно, **заключается в явном указании диапазона значений**, поддерживаемых intи System::Int32.

System::Int32 делает 32-разрядность более понятной для тех, кто читает код. Следует использовать intтам, где просто нужно "целое число", и использовать System::Int32там, где важен размер (криптографический код, структуры), чтобы читатели были в курсе этого, сохраняя то же самое.

Использование intили System::Int32результирующий код будут идентичны: разница заключается исключительно в явной читаемости или внешнем виде кода.

C# поддерживает обычный набор типов данных. Для каждого типа данных, который поддерживается C#, существует соответствующий тип .NET Common Language Runtime. Например, типу int соответствует тип System.Int32. Практически всегда можно использовать System.Int32 вместо int, но этого не рекомендуется делать, потому что в этом случае код становится более сложным для восприятия.  
  
Просто, double - это алиас класса Double, т.е. в коде пишем **double** тоже что new **System.Double**

Разницы нет, double - это просто псевдоним System.Double. То же самое касается int / Int32, float / Single и т.д.

Однако, как уже говорилось, разницы нет, однако, **D** ouble требует, чтобы вы добавили требуемое пространство имен при его использовании, тогда как double - это ключевое слово языка, которое позже переводится в System.Double, поэтому нет необходимости добавлять его.

21.Для чего используется тип dynamic?

Это ключевое слово позволяет опустить проверку типов во время компиляции. Кроме того, объекты, объявленные как dynamic, могут в течение работы программы **менять свой** тип.

22.В чем заключается главное отличие между var и dynamic?

**Первое отличие** между ключевыми словами **var** и **dynamic** заключается в том, когда определяется тип переменной. В случае ключевого слова var это будет первое присваивание значения переменной, обозначенной ключевым словом **var**. После этого присваивания изменить тип **значения переменной нельзя**. Однако в случае ключевого слова **dynamic**, **тип переменной можно изменять после первого присваивания**

**Вторым отличием** является то, что во время создания кода на переменных типа **dynamic** можно вызывать различные методы и свойства. Такой код будет скомпилирован без проблем, даже если мы будем знать, что фактический тип переменной в типе **dynamic** не поддерживает данный метод или свойство

23.Что такое неявно типизированная переменная?

 компилятору предоставляется возможность самому определить тип локальной переменной, исходя из значения, которым она инициализируется. Такая переменная называется ***неявно типизированной***.

Неявно типизированная переменная объявляется с помощью ключевого слова **var** и должна быть непременно инициализирована.



24.Для чего используют Nullable тип?

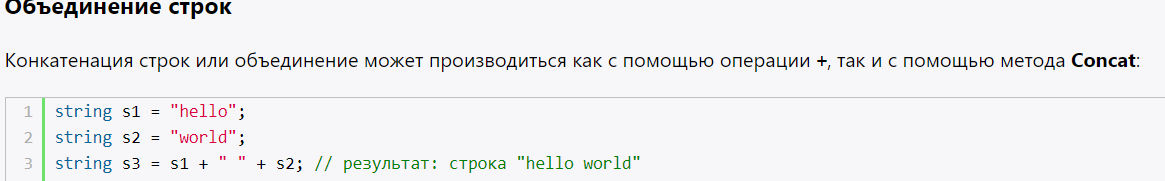
Чтобы присвоения переменной или параметру значимого типа значения **null**, эти переменная/параметр значимого типа должны представлять тип **nullable**



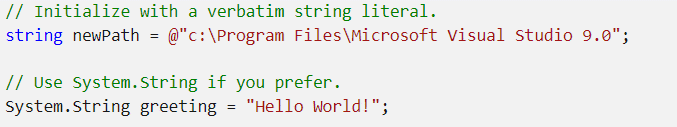
25.Как объявить строковый литерал? Какие операции можно выполнять со строкой?

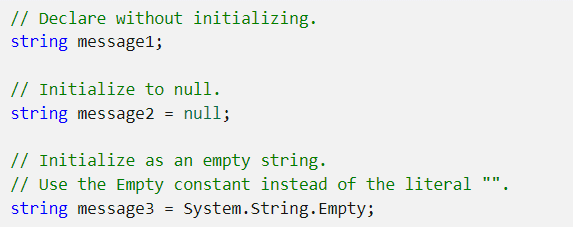
Строковые литералы представляют строки. Строки заключаются в двойные кавычки “HELLOW”

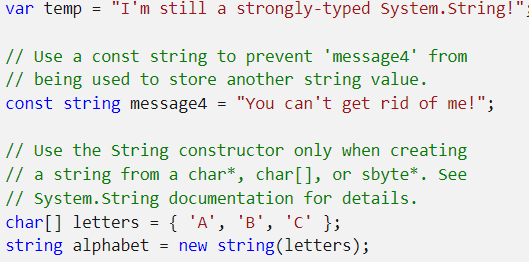
Операции:???????????????????



26.Какие есть способы для задания и инициализации строк?







27.Какие методы есть у типа String?

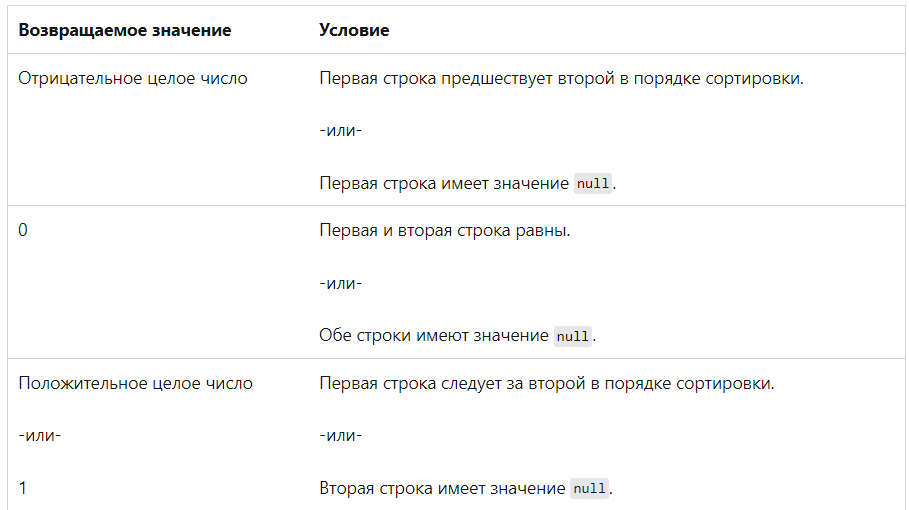
* **Compare**: сравнивает две строки с учетом текущей культуры (локали) пользователя
* **CompareOrdinal**: сравнивает две строки без учета локали
* **Contains**: определяет, содержится ли подстрока в строке
* **Concat**: соединяет строки
* **CopyTo**: копирует часть строки, начиная с определенного индекса в массив
* **EndsWith**: определяет, совпадает ли конец строки с подстрокой
* **Format**: форматирует строку
* **IndexOf**: находит индекс первого вхождения символа или подстроки в строке
* **Insert**: вставляет в строку подстроку
* **Join**: соединяет элементы массива строк
* **LastIndexOf**: находит индекс последнего вхождения символа или подстроки в строке
* **Replace**: замещает в строке символ или подстроку другим символом или подстрокой
* **Split**: разделяет одну строку на массив строк
* **Substring**: извлекает из строки подстроку, начиная с указанной позиции
* **ToLower**: переводит все символы строки в нижний регистр
* **ToUpper**: переводит все символы строки в верхний регистр
* **Trim**: удаляет начальные и конечные пробелы из строки

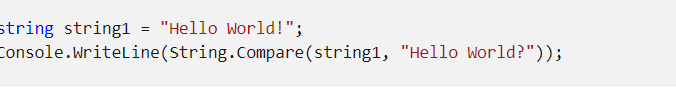
28.В чем отличие пустой и null строки?

**null** - это отсутствие какого-либо значения. Если вы попытаетесь обратиться к членам экземпляра **s1**, получите **NullReferenceException** (По сути, это даже и не экземпляр, просто переменная). Вторая строка, хоть и пустая, но память под неё в куче выделена, и вы можете к ней обращаться

29.Как можно выполнить сравнение строк?

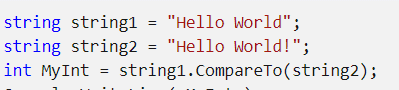
## Метод Compare



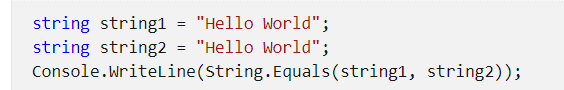


Метод [String.CompareOrdinal](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compareordinal) сравнивает два строковых объекта без учета локального языка и региональных параметров

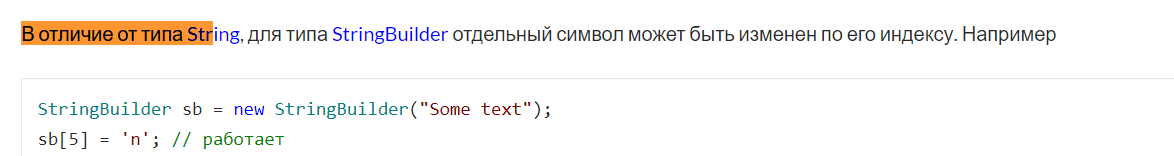
Метод [String.CompareTo](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compareto) сравнивает строку, которую инкапсулирует текущий строковый объект, с другой строкой или объектом.

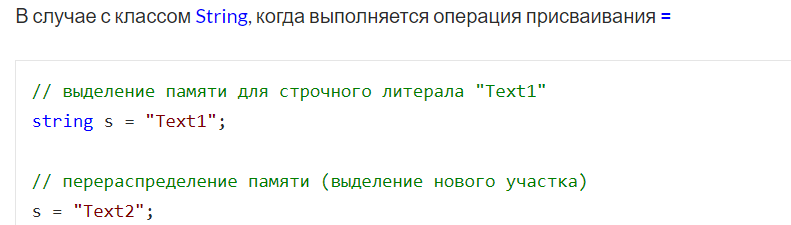


С помощью метода [String.Equals](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.equals) можно легко определить идентичность двух строк. Этот метод учитывает регистр и возвращает логическое значение true или false



30.В чем отличие типов String и StringBuilder?





**память** для нового строчного литерала («Text2») **перераспределяется по новому**: выделяется новый участок памяти, в который копируется новая измененная строка. Память, выделенная под предыдущий фрагмент, в дальнейшем уничтожается сборщиком мусора (Garbage collector).

В классе StringBuilder при изменении строки-оригинала, **память для новой строки выделяется только в случае, когда размер новой строки больше размера строки-оригина**ла. При этом, памяти выделяется в два раза больше предыдущего размера. Если размер новой строки меньше или равен размеру строки-оригинала, то память не перераспределяется. Такой механизм позволяет повысить быстродействие за счет уменьшения интенсивности (количества) выделения новых ресурсов и освобождения использованных ресурсов.

31.Поясните явные преобразования переменных с помощью команд Convert.

**Convert**содержит набор методов вида **ToType**, где вместо Type стоит системное название целевого типа значения (например *ToChar, ToInt32, ToBoolean*). Эти методы позволяют выполнять все возможные преобразования базовых типов C#.

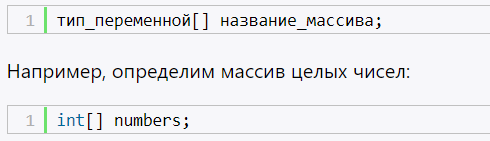
**Метод Convert.ChangeType преобразует любой объект в любой указанный тип.** При несоответствии типов или переполнении метод выдает исключение.

32.Как выполнить консольный ввод/вывод?

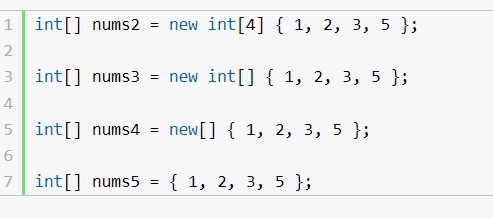




33.Приведите примеры определения и инициализации одномерных и двумерных массивов.

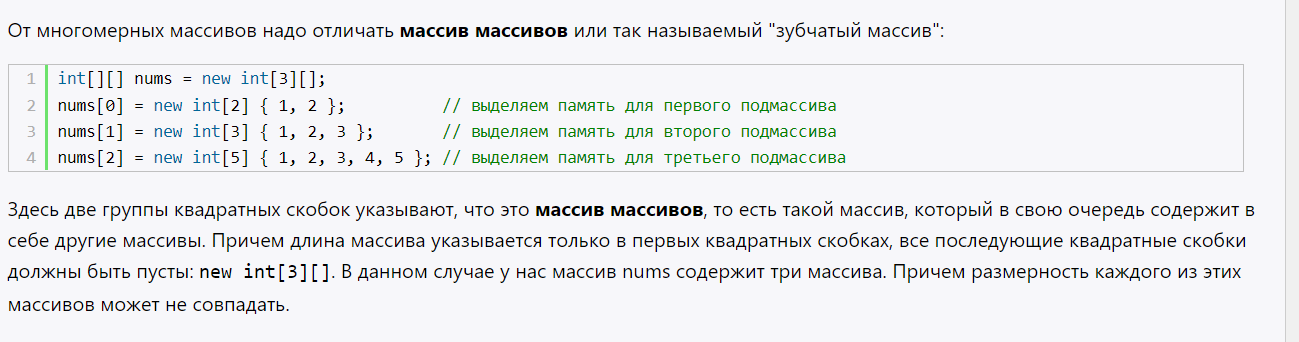


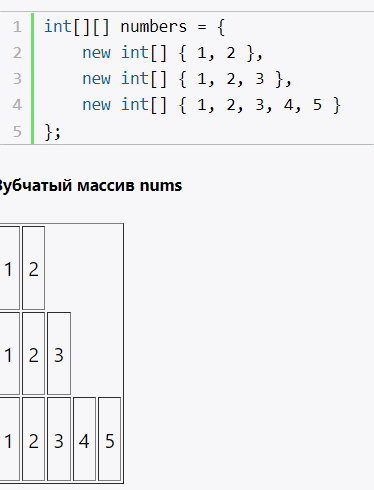
операцию new, мы выделили память для 4 элементов массива: new int[4]



 многомерный

34.Что такое ступенчатый массив? Как его задать?

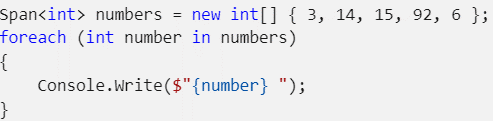




35.Какие типы можно использовать в foreach? Приведите пример.

Оператор foreach выполняет оператор или блок операторов для каждого элемента в экземпляре типа, который реализует интерфейс

Выполняет перебор всех элементов из множества в переменную



36.Что такое кортеж? Для чего и как он используется?

Основное предназначение кортежей - обобщение нескольких элементов в структуру с упрощенным синтаксисом

Кортежи предоставляют удобный способ для работы с набором значений, который был добавлен в версии C# 7.0.

Кортеж представляет набор значений, заключенных в круглые скобки



37.Что такое локальная функция? Какова область ее видимости?

Локальные функции представляют функции, определенные внутри других методов. Локальная функция, как правило, содержит действия, которые применяются только в рамках ее метода.

 При этом неважно, определена локальная функция до или после использования. Но вне ее метода локальная функция не может использоваться.(SUM)

void Compare(int[] numbers1, int[] numbers2)

{

    int numbers1Sum = Sum(numbers1);

    int numbers2Sum = Sum(numbers2);

    if (numbers1Sum > numbers2Sum)

        Console.WriteLine("сумма чисел из массива numbers1 больше");

    else

        Console.WriteLine("сумма чисел из массива numbers2 больше");

**int Sum(int[] numbers)**

**{**

**int result = 0;**

**foreach (int number in numbers)**

**result += number;**

**return result;**

**}**

}

int[] numbers1 = { 1, 2, 3 };

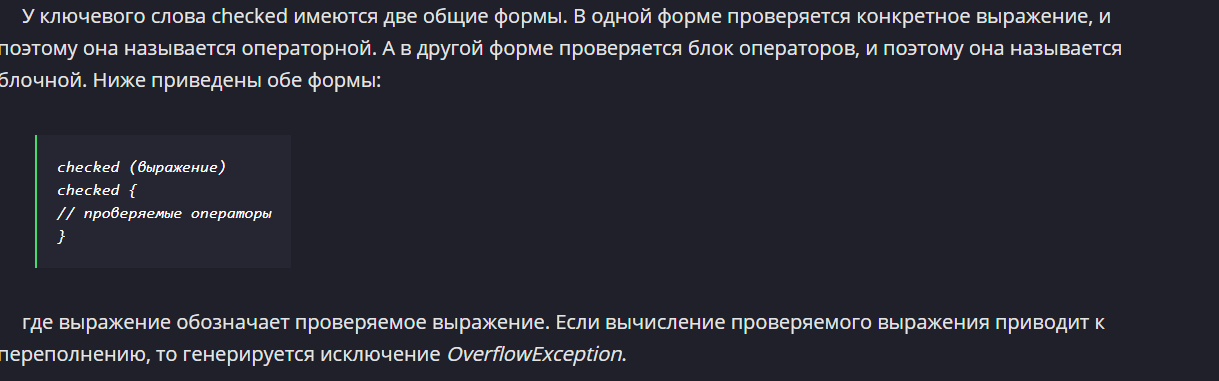
int[] numbers2 = { 3, 4, 5, 6, 7 };

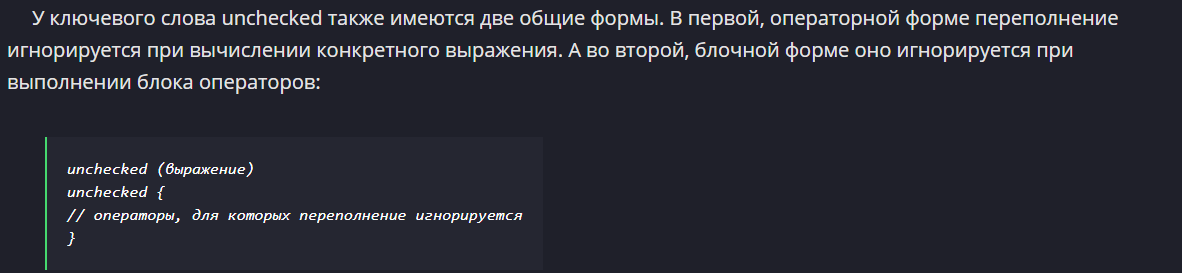
Compare(numbers1, numbers2);

38. В чем разница между кодом, заключенным в блок checked и кодом, заключенным в блок unchecked?

специальное средство, связанное с генерированием исключений, возникающих при переполнении в арифметических вычислениях. Как вам должно быть уже известно, результаты некоторых видов арифметических вычислений **могут превышать диапазон представления чисел для типа данных**, используемого в вычислении. В этом случае происходит так называемое ***переполнение****.*

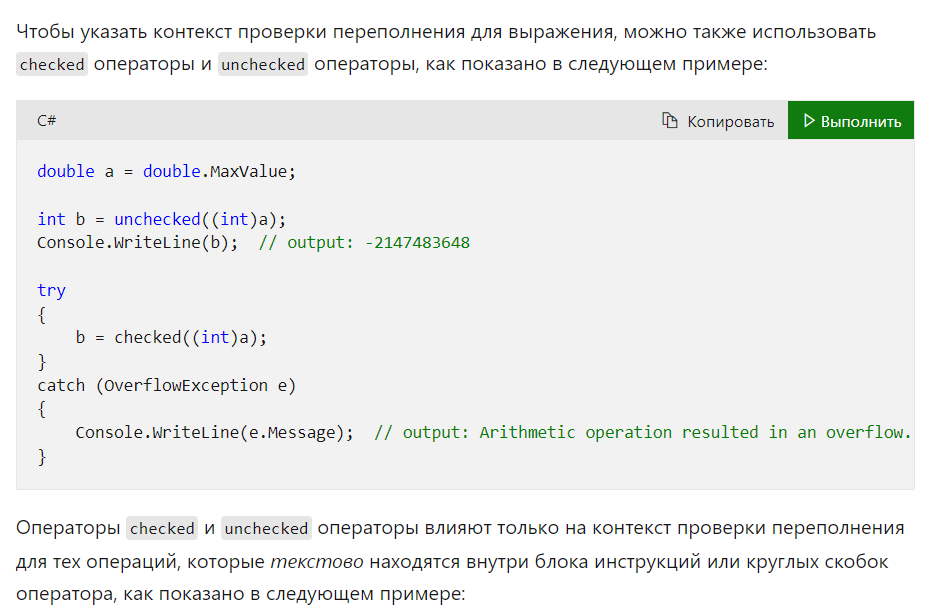
 будет ли в коде сгенерировано исключение при переполнении, с помощью ключевых словcheckedиunchecked





39.Какой контекст (checked/unchecked) применяется по умолчанию? Как можно переопределить это поведение?

По умолчанию: ператоры checked и unchecked операторы влияют только на контекст проверки переполнения для тех операций, которые текстово находятся внутри блока инструкций или круглых скобок оператора, как показано в следующем примере:



40.Для чего используется ключевое слово fixed? Каковы особенности его использования?

Ключевое fixed слово позволяет закрепить локальный элемент в стеке, чтобы предотвратить его сбор или перемещение во время сборки мусора. Он используется для низкоуровневых сценариев программирования.

fixed нужен для того, чтобы сборщик мусора не перетащил объект, на который указывает указатель в другое место

. Фиксированный оператор отключает сборщик мусора, чтобы переместить подвижную переменную. Фиксированный оператор может появиться только в небезопасном контексте. Фиксированный также может быть использован для создания буфера фиксированного размера.  
2, Фиксированный оператор устанавливает указатель на управляемом переменный и «пиннинг» переменный во время заявления. Если нет Фиксированного заявления, указатель на подвижном управляемом переменного не мал, так как восстановление мусора не может предсказать переменное. C # компилятор допускает только указатель на размещенную переменную для назначения в Fixed заявлении.  
3. После выполнения кода в среднем, любая фиксированная переменная освобождается и зависит от сбора мусора. Поэтому, не указывают на эти переменные за пределами фиксированного заявления.

