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

|  |  |
| --- | --- |
| Платформа .NET Framework состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET Framework. Основой платформы .NET Framework является среда CLR. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом средой накладываются условия строгой типизации и другие виды проверки точности кода обеспечивающие безопасность и надежность. Фактически основной  , | |
| задачей среды выполнения является управление кодом. |  |

1. Что такое CLR, FCL/BCL, CLI, IL?

-Общеязыковая исполнительная среда CLR решает задачи автоматического выявления типов .NET, загрузки этих типов и управление ними. Среда CLR осуществляет управление памятью, обслуживание приложения, обработку потоков и реализует многочисленные проверки связанные с безопасностью.

* CLS (Спецификация общего языка)

CLS - это подмножество CTS, оно определяет набор правил и ограничений, которым должен следовать каждый язык, работающий в среде .NET.

* Библиотека базовых классов (BCL)

Обозначает библиотеку базовых классов, также известную как библиотека классов (CL). BCL - это подмножество библиотеки классов Framework (FCL). Библиотека классов - это набор многоразовых типов, которые тесно интегрированы с

CLR. Библиотека базовых классов предоставляет классы и типы, которые полезны при выполнении повседневных операций, например, при работе со строковыми и примитивными типами, подключением к базе данных, операциями ввода-вывода.

* **IL** (Intermediate Language) — код на специальном языке, напоминающим ассемблер, но написанном для .NET. В него преобразуется код из других языков верхнего уровня (c#, VisualBasic). Вот тогда-то и пропадает зависимость от выбранного языка.\
* **Интерфейс командной строки** [(англ.](https://ru.bmstu.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) *Command line interface, CLI*) — разновидность текстового интерфейса(TUI) между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (*команд*), в UNIX-системах возможно применение мыши. Также известен под названием *консоль*.

1. Пояснить работу JIT-компилятора?

Компилятор JIT запускается **после** запуска программы и компилирует код (обычно байт-код или какие-то инструкции VM) на лету (или just-in-time, как это называется) в форму, которая обычно быстрее, как правило, собственный набор инструкций хоста CPU. A JIT имеет доступ к динамической информации о времени выполнения, в то время как стандартный компилятор этого не делает и может выполнять лучшие оптимизации, такие как часто используемые встроенные функции. JIT-

компиляция может быть применена как ко всей программе, так и к её отдельным частям. Например, текстовый редактор может на лету компилировать регулярные выражения для более быстрого поиска по тексту.

1. Что такое CTS (Common Type System)?

**Common Type System** (сокр. CTS, [рус.](https://ru.wikipedia.org/wiki/%D0%A0%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Общая система типов*) — часть [.NET Framework,](https://ru.wikipedia.org/wiki/.NET_Framework) формальная спецификация, определяющая, как какой-либо тип (класс, интерфейс, структура, встроенный тип данных) должен быть определён для его правильного выполнения средой .NET. Кроме того, данный стандарт определяет, как определения [типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и специальные значения типов представлены в компьютерной памяти. Целью разработки CTS было обеспечение возможности программам, написанным на различных языках программирования, легко обмениваться информацией. Как это принято в [языках программирования,](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [тип](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) может быть описан как определение набора допустимых значений (например, «все целые от 0 до 10») и допустимых операций над этими значениями (например, сложение и вычитание).

1. Какие аспекты поведения определяет тип System.Object?
2. Что находится в mscorlib dll?

Mscorlib.ni.dll - это файл DLL в Windows. DLL файлы необходимы программам или расширениям веббраузера, потому что они содержат программный код, данные и ресурсы. Бесплатный форум с информацией о файлах может помочь вам разобраться является ли mscorlib.ni.dll вирусом, трояном, программой-шпионом, рекламой, которую вы можете удалить, или файл принадлежит системе Windows или приложению, которому можно доверять.

сборка *mscorlib.dll*. В этой сборке содержится большое количество базовых типов, охватывающих широкий спектр типичных задач

|  |  |
| --- | --- |
| программирования, а также базовых типов данных, применяемых во всех языках .NET. При построении .NET-решений доступ к этой конкретной | |
| сборке будет предоставляться автоматически. |  |

1. Что такое «сборка»? Из чего состоит сборка .NET?

|  |  |
| --- | --- |
| 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. |  |

Сборки имеют следующие составляющие:

|  |  |
| --- | --- |
| • | Манифест, который содержит метаданные сборки |
| • | Метаданные типов. Используя эти метаданные, сборка определяет местоположение типов в файле приложения, а также места размещения их в памяти |
| • | Собственно код приложения на языке MSIL, в который компилируется код C# |
| • | Ресурсы |

1. Какие виды сборок существуют?

|  |
| --- |
| Сборки бывают двух видов: *разделяемые* и *приватные*.  **Приватные сборки**  Это простейший тип сборок. Обычно они поставляются с определенным программным обеспечением и предназначены для применения только в его составе. Обычный сценарий получения приватной сборки — это когда приложение поставляется в виде исполняемой программы и множества библиотек, код которых может быть использован только этим приложением |

# Разделяемые сборки

Назначение разделяемых сборок — служить библиотеками общего применения, которые могут использоваться любым другим приложением. Поскольку любое другое приложение может получить доступ к разделяемой сборке, возникает необходимость в некоторых предосторожностях, чтобы исключить описанные ниже риски.

|  |
| --- |
|  |

9. Что такое assembly manifest?

*Манифест сборки(assembly manifest) - это внутренняя часть сборки, которая позволяет ей быть самоописанной.*10.Что такое GAC?

11.Чем managed code отличается от unmanaged code **управляемый код**

Управляемый код-это то, что создают компиляторы Visual Basic .NET и C#. Он работает в среде CLR (Common Language Runtime), которая, помимо прочего, предлагает такие службы, как сбор мусора, проверка типов во время выполнения и проверка ссылок **неуправляемый код**

Неуправляемый код компилируется прямо в машинный код. Таким образом, по этому определению весь код, скомпилированный традиционными компиляторами C/C++, равен 'unmanaged code'. Кроме того, поскольку он компилируется в машинный код, а не в промежуточный язык, он непереносим.

Никакого свободного управления памятью или чего-либо еще, что предоставляет CLR.

12.Как и для чего определен метод Main?

метод Main() — это главный метод программы, ее входная точка, является частью класса. Класс, в котором находится этот метод, по умолчанию называется Program. Это статический метод, имеет ключевое слово static, поэтому он реализуется еще до создания объекта класса, в котором он находится13.Варианты использования директивы using( using Directive ) в C#.

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.15.Что такое примитивные типы данных? Перечислите их.

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

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. 17.Какие типы относятся к типам-значениям?

Для объявления ссылочных типов используются следующие ключевые слова:

* [class](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/class)
* [interface](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/interface)
* [delegate](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/reference-types)

В C# также предусмотрены следующие встроенные ссылочные типы:

1. [dynamic](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/reference-types)
2. [object](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/reference-types)
3. [string](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/reference-types)

К типам значений относятся:

|  |  |
| --- | --- |
| * Целочисленные типы (*byte, sbyte, char, short, ushort, int, uint, long, ulong*); * Типы с плавающей запятой (*float, double*); * Тип *decimal*; * Тип *bool*; * Пользовательские структуры (*struct*); | |
| - Перечисления (*enum*). |  |

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

19.Что такое упаковка и распаковка значимых типов?

**При упаковке происходит следующее:**

* В управляемой куче выделяется память длиной значимого типа и двумя дополнительными полями, (необходимыми для всех объектов в управляемой куче: **указатель на объект-тип** и **SyncBlockIndex**).
* Значение **значимого типа** копируется в **кучю**
* Возвращается адрес объекта. Этот адрес является ссылкой на объект; **значимый тип** превратился в **ссылочный** **аспаковка** это преобразование **Object**типа в **value**(**значимый**) тип

**При распаковке происходит следующее:**

* Сначала извлекается адрес поля из упакованного объекта
* значения поля копируется из кучи в экземпляр значимого типа, находящийся в стеке

20.В чем заключается разница между int и System.Int32? double и System.Double и т.д.?

**int** - примитив, доступный в компиляторе C#, в то время как **Int32**является типом [**FCL**](https://en.wikipedia.org/wiki/Framework_Class_Library) и следовательно доступен для всех языков, соответствующих CLS.

|  |  |
| --- | --- |
| 10.01.2013, 23:44 | [**2**](https://www.cyberforum.ru/post3982612.html) |
| 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** | |

21.Для чего используется тип dynamic?

На самом деле это статический Тип, но в отличие от других статических типов, (потенциальные) члены динамического объекта не проверяются компилятором

|  |  |
| --- | --- |
| 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?

Переменные, объявленные с помощью var, неявно, но статически типизированы. Переменные, объявленные с помощью Динамические, динамически типизируются.

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** .

24.Для чего используют Nullable тип?

Они очень полезны при работе со значениями из базы данных, когда значение, возвращаемое из вашей таблицы, равно NULL . Представьте себе целое значение в таблице базы данных, которое может быть NULL, такое может быть представлено только с 0 , если переменная c# не является nullable-регулярным целым числом

Представьте себе, что существует целочисленная переменная id , представляющая конкретный id .

Вы можете хранить 1, 44 или что-нибудь еще. Но что делать, если вы не знаете удостоверение личности. Вы не можете просто хранить -1 или 0. Возможно, вы подумываете назначитьnull , но обычно null не может быть присвоено типам значений.

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

Console.WriteLine("hello");

// Declare without initializing.

string message1;

// Initialize to null.

string message2 = null;

// Initialize as an empty string.

// Use the Empty constant instead of the literal "".

string message3 = System.String.Empty;

// Initialize with a regular string literal.

string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.

string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.

System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)

// you can use implicit typing.

var temp = "I'm still a strongly-typed System.String!";

// Use a const string to prevent 'message4' from

// being used to store another string value.

const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating

// a string from a char\*, char[], or sbyte\*. See

// System.String documentation for details.

char[] letters = { 'A', 'B', 'C' };

string alphabet = new string(letters);

СРАВНЕНИЕ СТРОК

Порядковые сравнения по умолчаию

-[String.CompareTo](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compareto)

-[String.Equals](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.equals)

-[String.Equality](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.op_equality) и [String.Inequality](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.op_inequality),т.е.операторы [равенства  == и !=](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/equality-operators#string-equality) соответствено

string root = @"C:\users";

string root2 = @"C:\Users";

bool result = root.Equals(root2);

Console.WriteLine($"Ordinal comparison: <{root}> and <{root2}> are {(result ? "equal." : "not equal.")}");

result = root.Equals(root2, StringComparison.Ordinal);

Console.WriteLine($"Ordinal comparison: <{root}> and <{root2}> are {(result ? "equal." : "not equal.")}");

Console.WriteLine($"Using == says that <{root}> and <{root2}> are {(root == root2 ? "equal" : "not equal")}");

**Порядковые сравнения без учета регистра**

Метод [String.Equals(String, StringComparison)](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.equals" \l "System_String_Equals_System_String_System_StringComparison_) позволяет указать значение [StringComparison](https://docs.microsoft.com/ru-ru/dotnet/api/system.stringcomparison) для объекта [StringComparison.OrdinalIgnoreCase](https://docs.microsoft.com/ru-ru/dotnet/api/system.stringcomparison" \l "System_StringComparison_OrdinalIgnoreCase) для порядкового сравнения без учета регистра. Также имеется статический метод [String.Compare(String, String, StringComparison)](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compare" \l "System_String_Compare_System_String_System_String_System_StringComparison_), позволяющий проводить порядковое сравнение без учета регистра, если указать значение [StringComparison.OrdinalIgnoreCase](https://docs.microsoft.com/ru-ru/dotnet/api/system.stringcomparison" \l "System_StringComparison_OrdinalIgnoreCase) для аргумента [StringComparison](https://docs.microsoft.com/ru-ru/dotnet/api/system.stringcomparison). Это показано в следующем коде:

string root = @"C:\users";

string root2 = @"C:\Users";

bool result = root.Equals(root2, StringComparison.OrdinalIgnoreCase);

bool areEqual = String.Equals(root, root2, StringComparison.OrdinalIgnoreCase);

int comparison = String.Compare(root, root2, comparisonType: StringComparison.OrdinalIgnoreCase);

Console.WriteLine($"Ordinal ignore case: <{root}> and <{root2}> are {(result ? "equal." : "not equal.")}");

Console.WriteLine($"Ordinal static ignore case: <{root}> and <{root2}> are {(areEqual ? "equal." : "not equal.")}");

if (comparison < 0)

Console.WriteLine($"<{root}> is less than <{root2}>");

else if (comparison > 0)

Console.WriteLine($"<{root}> is greater than <{root2}>");

else

Console.WriteLine($"<{root}> and <{root2}> are equivalent in order");

**Лингвистические сравнения**

string first = "Sie tanzen auf der Straße.";

string second = "Sie tanzen auf der Strasse.";

Console.WriteLine($"First sentence is <{first}>");

Console.WriteLine($"Second sentence is <{second}>");

bool equal = String.Equals(first, second, StringComparison.InvariantCulture);

Console.WriteLine($"The two strings {(equal == true ? "are" : "are not")} equal.");

showComparison(first, second);

string word = "coop";

string words = "co-op";

string other = "cop";

showComparison(word, words);

showComparison(word, other);

showComparison(words, other);

void showComparison(string one, string two)

{

int compareLinguistic = String.Compare(one, two, StringComparison.InvariantCulture);

int compareOrdinal = String.Compare(one, two, StringComparison.Ordinal);

if (compareLinguistic < 0)

Console.WriteLine($"<{one}> is less than <{two}> using invariant culture");

else if (compareLinguistic > 0)

Console.WriteLine($"<{one}> is greater than <{two}> using invariant culture");

else

Console.WriteLine($"<{one}> and <{two}> are equivalent in order using invariant culture");

if (compareOrdinal < 0)

Console.WriteLine($"<{one}> is less than <{two}> using ordinal comparison");

else if (compareOrdinal > 0)

Console.WriteLine($"<{one}> is greater than <{two}> using ordinal comparison");

else

Console.WriteLine($"<{one}> and <{two}> are equivalent in order using ordinal comparison");

}

СКРЕКА СТРОК

Для сцепки строковых переменных вы можете использовать операторы + или +=, [интерполяцию строк](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/tokens/interpolated), а также методы [String.Format](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.format), [String.Concat](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.concat), [String.Join](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.join) или [StringBuilder.Append](https://docs.microsoft.com/ru-ru/dotnet/api/system.text.stringbuilder.append).

string userName = "<Type your name here>";

string dateString = DateTime.Today.ToShortDateString();

// Use the + and += operators for one-time concatenations.

string str = "Hello " + userName + ". Today is " + dateString + ".";

System.Console.WriteLine(str);

str += " How are you today?";

System.Console.WriteLine(str);

Разделение строк

Метод String.Split создает массив подстрок, разбивая входную строку по одному или нескольким разделителям.

string phrase = "The quick brown fox jumps over the lazy dog.";

string[] words = phrase.Split(' ');

foreach (var word in words)

{

System.Console.WriteLine($"<{word}>");

}

интерполяция строк

Интерполированная строка — это строковый литерал, который может содержать выражения интерполяции.

Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, it's {date:HH:mm} now.");  
26.Какие есть способы для задания и инициализации строк?

// Declare without initializing.

string message1;

// Initialize to null.

string message2 = null;

// Initialize as an empty string.

// Use the Empty constant instead of the literal "".

string message3 = System.String.Empty;

// Initialize with a regular string literal.

string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.

string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.

System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)

// you can use implicit typing.

var temp = "I'm still a strongly-typed System.String!";

// Use a const string to prevent 'message4' from

// being used to store another string value.

const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating

// a string from a char\*, char[], or sbyte\*. See

// System.String documentation for details.

char[] letters = { 'A', 'B', 'C' };

string alphabet = new string(letters);

27.Какие методы есть у типа String?

### Основные методы класса String

* **concat()**: объединяет строки
* **valueOf()**: преобразует объект в строковый вид
* **join()**: соединяет строки с учетом разделителя
* **сompare()**: сравнивает две строки
* **charAt()**: возвращает символ строки по индексу
* **getChars()**: возвращает группу символов
* **equals()**: сравнивает строки с учетом регистра
* **equalsIgnoreCase()**: сравнивает строки без учета регистра
* **regionMatches()**: сравнивает подстроки в строках
* **indexOf()**: находит индекс первого вхождения подстроки в строку
* **lastIndexOf()**: находит индекс последнего вхождения подстроки в строку
* **startsWith()**: определяет, начинается ли строка с подстроки
* **endsWith()**: определяет, заканчивается ли строка на определенную подстроку
* **replace()**: заменяет в строке одну подстроку на другую
* **trim()**: удаляет начальные и конечные пробелы
* **substring()**: возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса
* **toLowerCase()**: переводит все символы строки в нижний регистр
* **toUpperCase()**: переводит все символы строки в верхний регистр
* метод **length()** для нахождения длины строки или длины набора символов:

|  |  |
| --- | --- |
| 1  2 | String str1 = "Java";  System.out.println(str1.length()); // 4 |

**toCharArray()** можно обратно преобразовать строку в массив символов:

|  |  |
| --- | --- |
| 1  2 | String str1 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});  char[] helloArray = str1.toCharArray(); |

позволяет проверить строку на пустоту - **isEmpty()**. Если строка пуста, он возвращает true:

|  |  |
| --- | --- |
| 1  2 | String s = "";   // строка не указывает на объект  if(s.length() == 0) System.out.println("String is empty"); |

28.В чем отличие пустой и null строки?

$a = null;  
$a = '';

в обоих случаях у а есть вполне конкретные значения и тип.  
[**null**](http://php.net/manual/ru/language.types.null.php)  
[**string**](http://php.net/manual/ru/language.types.string.php)

NULL - это тип данных, который используется для обозначения пустой переменной. а $a='' - это тип данных строка, пустая, но строка, а не пустота.

Null ничего не значит. Это буквально. Null - значение ссылочной переменной. Но пустая строка пуста. Она дает length=0. Пустая строка - это пустое значение, означает, что в строке ничего нет.

29.Как можно выполнить сравнение строк?

В C# правильно сравнивать строки и через ==, и через Equals. Но более предпочтительным будет сравнивать через ==.

операторы != и == для сравнения.

Если же необходимо сравнить строки по ссылке, это следует делать так: if ((Object) s1 == (Object) s2) {}.

Console.WriteLine(s1 == s4);   
        Console.WriteLine((Object) s1 == (Object) s3);

30.В чем отличие типов String и StringBuilder?

Основное отличие - StringBuilder можно изменять после создания.

Строка иммьютабельна (её невозможно изменить после того, как она создана). Если вы что-то прибавляете к строке, то в действительности вы не получите старую строку + добавку, вы получите новую строку, составленную из старой и добавки. Отсюда все проблемы с производительностью в приложениях, которые последовательно формируют большие строки.  
  
StringBuilder же будет последовательно выделять память и "по-настоящему" дозаписываться данными. Так что при построении больших строк - только StringBuilder.

String - один из двух базовых классов, как и тип object. Любой объект можно привести к .ToString() как и любой класс к объекту(для крайних случаев есть перегрузка метода .ToString()).  
StringBuilder - в основном используется в потоках чтения-записи

31.Поясните явные преобразования переменных с помощью команд Convert.

способ преобразования данных с помощью класса **Convert**, в котором есть много статических методов (с префиксом **To**). Данные статические методы выполняют преобразования.

static void Main(string[] args)

{

    short s = 45;

    byte b = Convert.ToByte(s);

    Console.WriteLine("Byte = {0}", b);

    long l = 1234;

    int i = Convert.ToInt32(l);

    Console.WriteLine("Int = {0}", i);

    Console.ReadLine();

}

32.Как выполнить консольный ввод/вывод?

Для вывода информации на консоль - **Console.WriteLine**.

получать информацию с консоли - **Console.ReadLine()**. Он позволяет получить введенную строку.

Console.Write("Введите свое имя: ");

            string name = Console.ReadLine();

            Console.WriteLine($"Привет {name}");

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

Вы создаете одномерный массив, используя оператор new, определяющий тип элемента массива и количество элементов

int[] array = new int[5];

string[] stringArray = new string[6];

Инициализация массива

Вы можете инициализировать элементы массива при объявлении массива.

int[] array1 = new int[] { 1, 3, 5, 7, 9 };

string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

int[] array2 = { 1, 3, 5, 7, 9 };

string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

Массивы могут иметь более одного измерения. Например, следующее объявление создает двумерный массив из четырех строк и двух столбцов.

int[,] array = new int[4, 2];

Инициализация массива

Вы можете инициализировать массив после объявления, как показано в следующем примере.

// Two-dimensional array.

int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

// The same array with dimensions specified.

int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

// A similar array with string elements.

string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" },

{ "five", "six" } };

// Three-dimensional array.

int[, ,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },

{ { 7, 8, 9 }, { 10, 11, 12 } } };

// The same array with dimensions specified.

int[, ,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },

{ { 7, 8, 9 }, { 10, 11, 12 } } };

// Accessing array elements.

System.Console.WriteLine(array2D[0, 0]);

System.Console.WriteLine(array2D[0, 1]);

System.Console.WriteLine(array2D[1, 0]);

System.Console.WriteLine(array2D[1, 1]);

System.Console.WriteLine(array2D[3, 0]);

System.Console.WriteLine(array2Db[1, 0]);

System.Console.WriteLine(array3Da[1, 0, 1]);

System.Console.WriteLine(array3D[1, 1, 2]);

// Getting the total count of elements or the length of a given dimension.

var allLength = array3D.Length;

var total = 1;

for (int i = 0; i < array3D.Rank; i++) {

total \*= array3D.GetLength(i);

}

System.Console.WriteLine("{0} equals {1}", allLength, total);

// Output:

// 1

// 2

// 3

// 4

// 7

// three

// 8

// 12

// 12 equals 12

Вы также можете инициализировать массив без указания ранга.

int[,] array5;

array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } }; // OK

//array5 = {{1,2}, {3,4}, {5,6}, {7,8}}; // Error

34.Что такое ступенчатый массив? Как его задать?

Неровный массив - это массив, элементами которого являются массивы. Элементы зубчатого массива могут быть разных размеров и размеров. Неровный массив иногда называют «массивом массивов».

Перед использованием jaggedArrayего элементы должны быть инициализированы. Вы можете инициализировать элементы следующим образом:

1)

jaggedArray[0] = new int[5];

jaggedArray[1] = new int[4];

jaggedArray[2] = new int[2];

2)

jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };

jaggedArray[1] = new int[] { 0, 2, 4, 6 };

jaggedArray[2] = new int[] { 11, 22 };

3)

int[][] jaggedArray2 = new int[][]

{

new int[] { 1, 3, 5, 7, 9 },

new int[] { 0, 2, 4, 6 },

new int[] { 11, 22 }

};

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

Оператор цикла **foreach** в Си-шарп служит для перебора элементов коллекции. К коллекциям относятся массивы, списки List и пользовательские классы коллекций.

Оператор foreach выполняет оператор или блок операторов для каждого элемента в экземпляре типа, который реализует интерфейс System.Collections.IEnumerable или System.Collections.Generic.IEnumerable<T>, как показано в следующем примере.

foreach (тип имя\_переменной\_цикла in коллекция)

оператор;

var fibNumbers = new List<int> { 0, 1, 1, 2, 3, 5, 8, 13 };

int count = 0;

foreach (int element in fibNumbers)

{

count++;

Console.WriteLine($"Element #{count}: {element}");

}

Console.WriteLine($"Number of elements: {count}");

Оператор foreach не ограничен этими типами. Его можно использовать с экземпляром любого типа, который удовлетворяет следующим условиям:

тип содержит открытый метод GetEnumerator без параметров со следующим типом возвращаемого значения: класс, структура или тип интерфейса. Начиная с C# 9.0 метод GetEnumerator может быть методом расширения типа.

тип возвращаемого значения метода GetEnumerator должен содержать открытое свойство Current и открытый метод MoveNext без параметров с типом возвращаемого значения Boolean.

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

Кортежи предоставляют удобный способ для работы с набором значений

Кортеж представляет набор значений, заключенных в круглые скобки:

var tuple = (5, 10);

(int, int) tuple = (5, 10);

(string, int, double) person = ("Tom", 25, 81.23);

Кортежи могут передаваться в качестве параметров в метод, могут быть возвращаемым результатом функции, либо использоваться иным образом.

Например, одной из распространенных ситуаций является возвращение из функции двух и более значений, в то время как функция можно возвращать только одно значение. И кортежи представляют оптимальный способ для решения этой задачи:

static void Main(string[] args)

{

    var tuple = GetValues();

    Console.WriteLine(tuple.Item1); // 1

    Console.WriteLine(tuple.Item2); // 3

    Console.Read();

}

private static (int, int) GetValues()

{

    var result = (1, 3);

    return result;

}

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

Локальные функции представляют функции, определенные внутри других методов. Mожет быть вызвана только внутри этого метода

Локальные функции - это частные методы типа, вложенные в другой член. Их можно вызвать только из содержащего их члена. Локальные функции могут быть объявлены и вызваны из:

Методы, особенно методы итератора и асинхронные методы

Конструкторы

Аксессоры собственности

Аксессоры событий

Анонимные методы

Лямбда-выражения

Финализаторы

Другие локальные функции

Однако локальные функции нельзя объявлять внутри члена, воплощающего выражение.

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

uncheckedКлючевое слово используется для подавления переполнения-проверки для арифметических операций интегрального типа и преобразования.

checkedКлючевое слово используется для явного включения переполнения проверки арифметических операций интегрального типа и преобразования.

если требуется указать, что выражение будет проверяться на переполнение, следует использовать ключевое слово checked, а если требуется проигнорировать переполнение — ключевое слово unchecked. В последнем случае результат усекается, чтобы не выйти за пределы диапазона представления чисел для целевого типа выражения.

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

40.Для чего используется ключевое слово fixed?

Каковы особенности его использования?

fixedКлючевое слово позволяет «закрепить» локальную систему в стеке, чтобы предотвратить их сбор или перемещение во время сборки мусора. Он используется для сценариев низкого уровня программирования.

fixed имеет два применения:

* он позволяет закрепить массив и получить указатель на данные
* при использовании в поле unsafe struct он объявляет "fixed buffer"-зарезервированный блок пространства в типе, доступ к которому осуществляется через указатели, а не обычные поля

- Указатель фиксирован, пока находится в области видимости, и, когда он выходит за пределы области, он больше не является фиксированным. fixed не может использоваться вне контекста use привязки. Необходимо привязать указатель к имени с помощью use .

Использование метода fixed должно выполняться в выражении в функции или в методе. Его нельзя использовать в области действия уровня скрипта или модуля.

Как и все коды указателей, это незащищенная функция, которая выдает предупреждение при использовании.