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

Microsoft.NET (.NET Framework) – программная платформа. Содержит следующие основные компоненты:

►CLR (Common Language Runtime) – Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация)  
► FCL (.NET Framework Class Library) – соответствующая CLS спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений)

Платформа .NET Framework — это управляемая среда выполнения для ОС Windows, предоставляющая разнообразные службы выполняемым в ней приложениям. Она состоит из двух основных компонентов: среды CLR — механизма, управляющего выполняющимися приложениями, и библиотеки классов .NET Framework — библиотеки проверенного кода, предназначенного для повторного использования, который разработчики могут вызывать из своих приложений.

2. Что такое CLR, FCL/BCL, CLI, IL?

Common Language Runtime ([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) *CLR* — *общеязыковая исполняющая среда*) — исполняющая среда для [байт-кода](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) [CIL](https://ru.wikipedia.org/wiki/Common_Intermediate_Language) ([MSIL](https://ru.wikipedia.org/wiki/MSIL)), в который компилируются программы, написанные на [.NET-совместимых языках программирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_.NET-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2)

**FCL (.NET Framework Class Library)** – соответствующая CLS спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений)

Библиотека базового класса (BCL) буквально такова, база. Он содержит основные, фундаментальные типы, такие как System.String и System.DateTime.

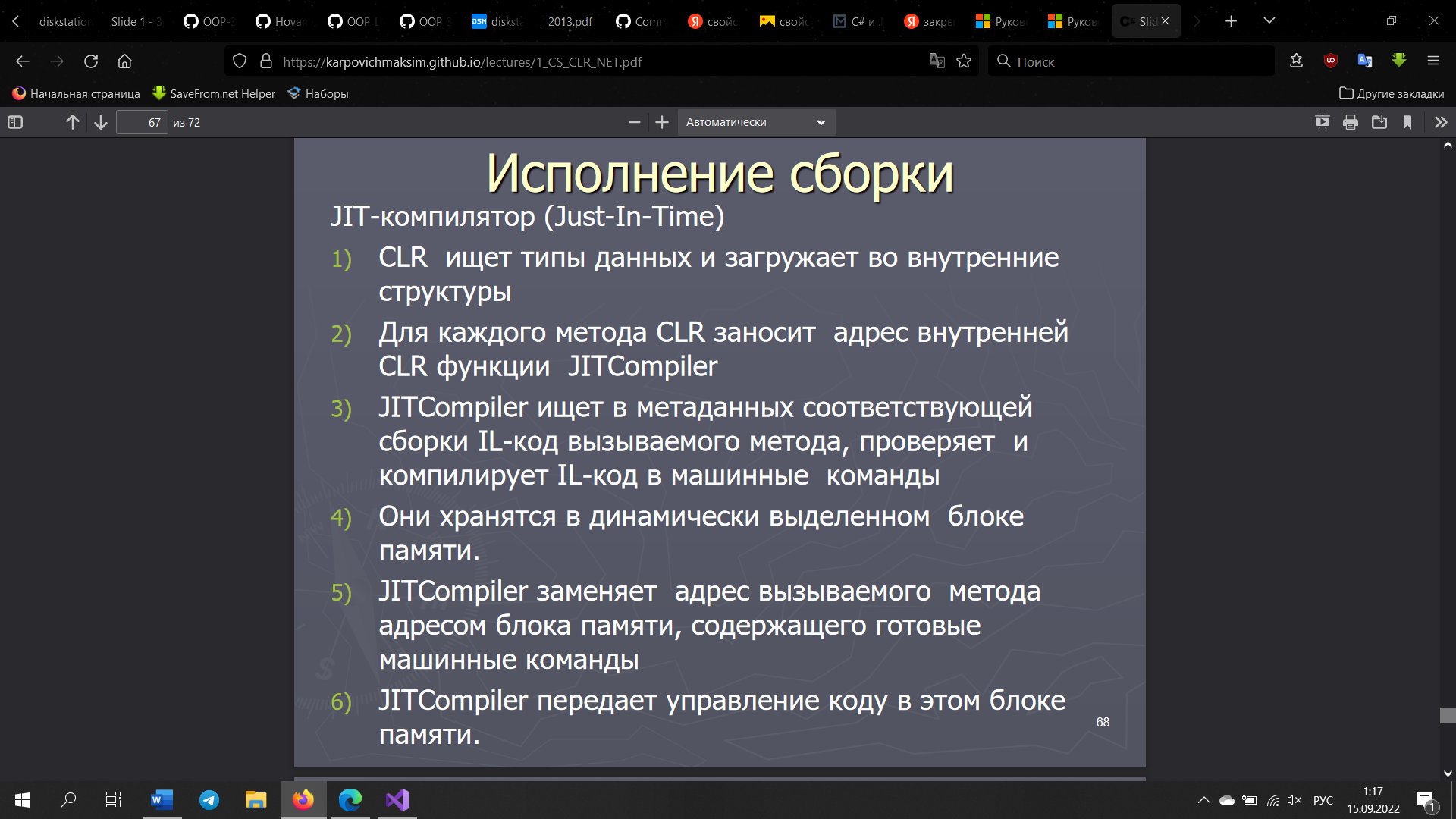
Библиотека классов классов (FCL) - это более широкая библиотека, которая содержит всю совокупность: ASP.NET, WinForms, стек XML, ADO.NET и другие. Вы можете сказать, что FCL включает BCL.

**CLI (Common Language Infrastructure)** – спецификация общеязыковой инфраструктуры. Определяет архитектуру исполнительной системы и набор представляемых сервисов.

Язык CIL также нередко называют просто IL от [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) ***i****ntermediate****l****anguage* — буквально «промежуточный язык».

все CLR-совместимые компиляторы генерируют IL-код. (Подробнее об IL-коде рассказано далее в этой главе.) IL-код иногда называют управляемым (managed code), потому что CLR управляет его выполнением

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.

JIT — это Just-inTime, что подразумевает компиляцию кода, когда это нужно, а не до выполнения программы. Поэтому JIT обладает следующими свойствами:

* не весь код сразу компилируется в машинный, а только необходимая часть, — это снижает нагрузку на процессор;
* машинный код генерируется во время выполнения программы, значит, он будет изначально оптимизирован под архитектуру устройства, что позволит делать кроссплатформенные программы.

4. Что такое CTS (Common Type System)?

Система общих типов определяет, как типы объявляются, используются и управляются в общеязыковой среде выполнения, а также является важной частью поддержки среды выполнения для межъязыковой интеграции.

Система общих типов выполняет следующие функции:

Устанавливает структуру, которая помогает обеспечить межъязыковую интеграцию, безопасность типов и высокопроизводительное выполнение кода.

Предоставляет объектно-ориентированную модель, поддерживающую полную реализацию многих языков программирования.

Определяет правила, которым должны следовать языки, что помогает гарантировать, что объекты, написанные на разных языках, могут взаимодействовать друг с другом.

Предоставляет библиотеку, содержащую примитивные типы данных (такие как Boolean , Byte , Char , Int32 и UInt64 ), используемые при разработке приложений.

5. Какие аспекты поведения определяет тип System.Object?

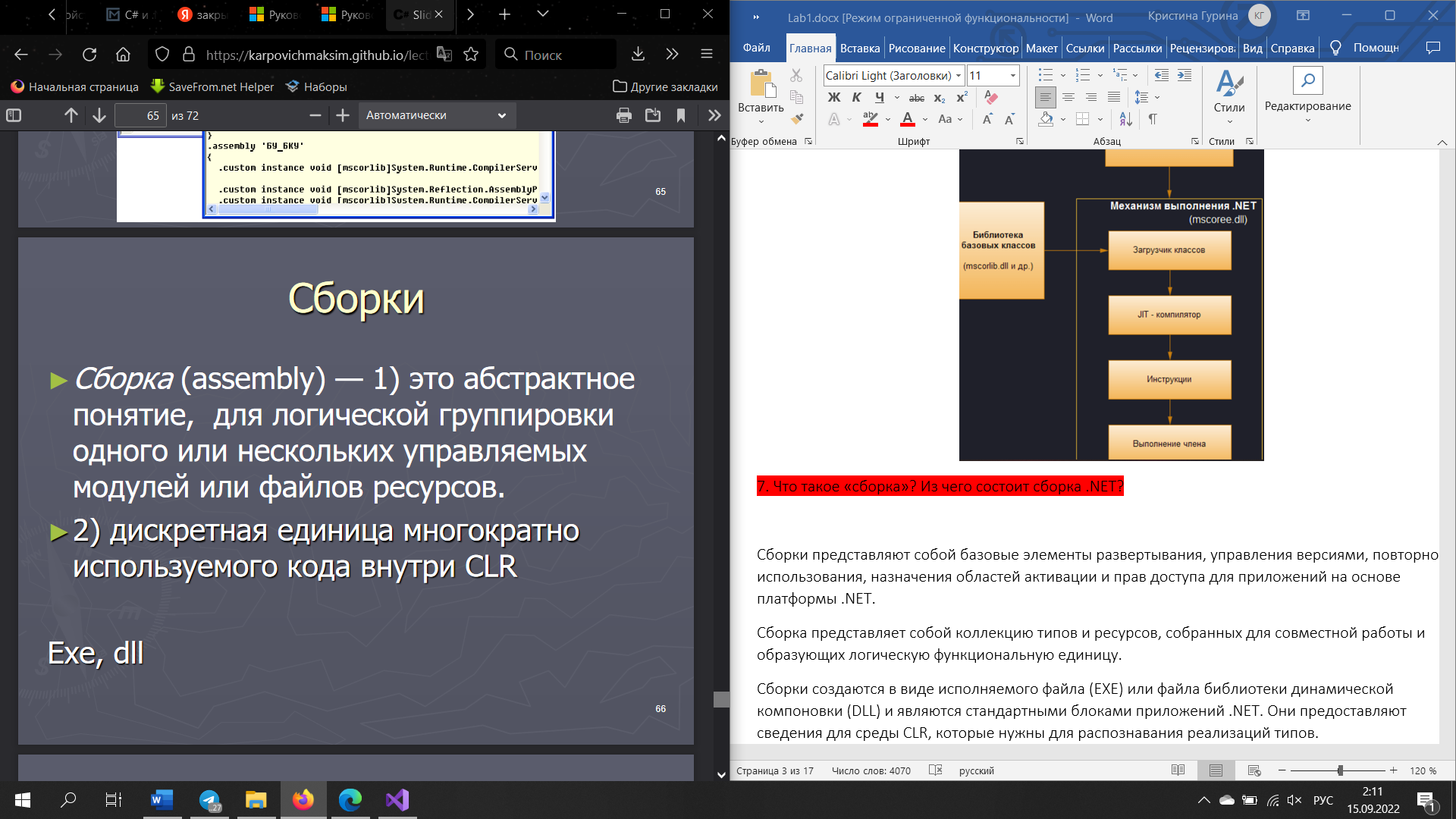
определяет способ объявления, использования и управления типами в среде CLR

6. Что находится в mscorlib dll?

Хотя вся библиотека базовых классов поделена на ряд отдельных сборок, главной среди них является 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.



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

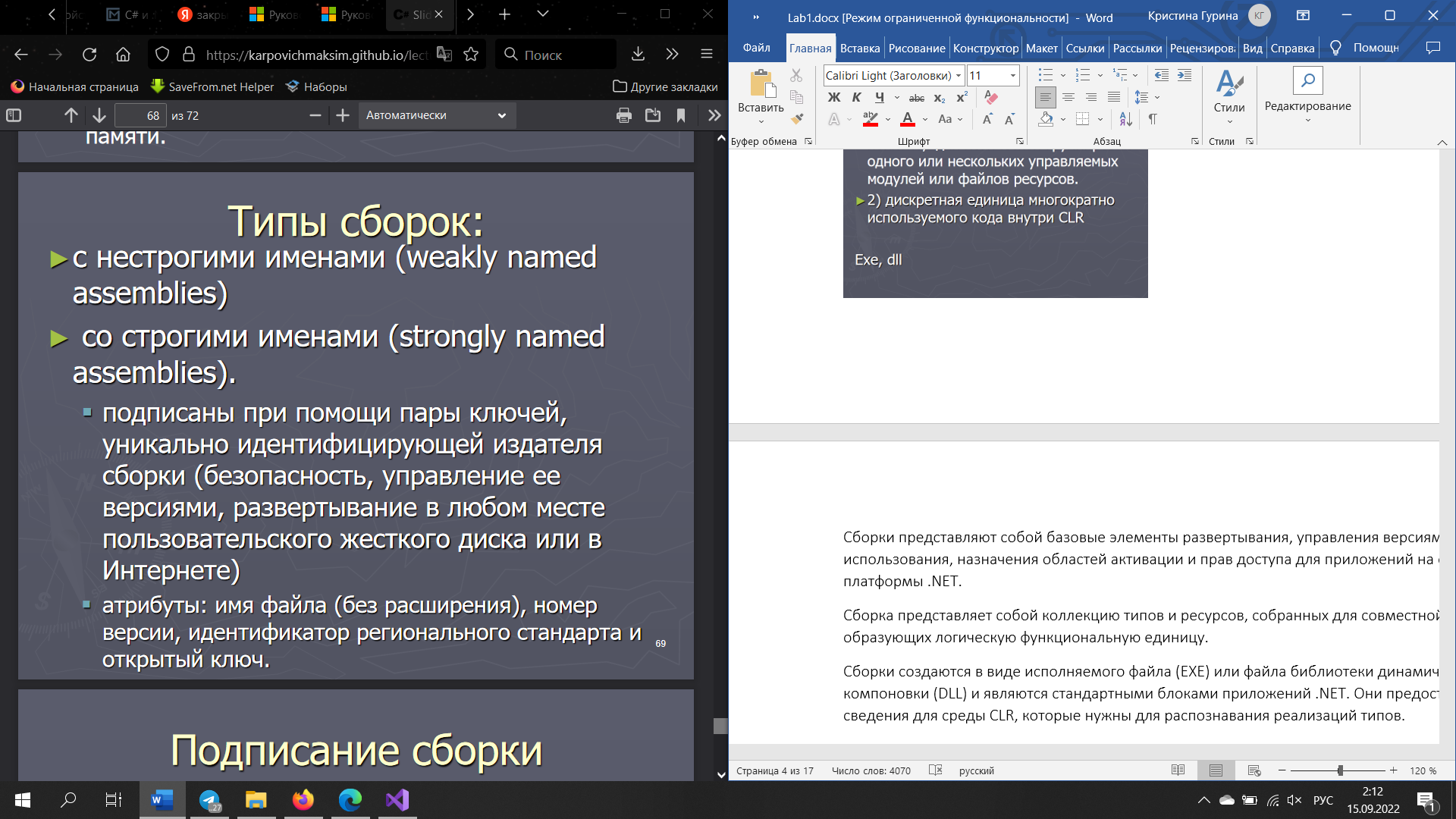


Сборки представляют собой базовые элементы развертывания, управления версиями, повторного использования, назначения областей активации и прав доступа для приложений на основе платформы .NET.

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.

8. Какие виды сборок существуют?



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.

Система гарантирует, что приватные сборки не будут применяться другим программным обеспечением, потому что приложение может загружать только приватные сборки, находящиеся в той же папке, где хранится и главная исполняемая программа, которая их загружает, либо во вложенных папках.

Поскольку приватные сборки могут быть использованы только тем пакетом программного обеспечения, для которого они предназначены, имеется возможность управлять тем программным обеспечением, которое их использует. Отсюда уменьшается необходимость в некоторых предосторожностях для обеспечения безопасности, поскольку нет риска, например, того, что какое-то коммерческое программное обеспечение перепишет сборки их новыми версиями (за исключением случаев, когда программное обеспечение специально разработано для нанесения определенного вреда). Не существует также проблемы [коллизии имен](https://professorweb.ru/my/glossariy/glossy/files/100.php#k1). Если окажется, что классы в приватной сборке имеют те же имена, что и классы в чьей-то чужой приватной сборке, то это не имеет значения, поскольку каждое приложение будет в состоянии видеть только один набор собственных приватных сборок.

## 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.

* Коллизия имен, когда разделяемая сборка, поставленная другой компанией, реализует типы с теми же именами, что используются в вашей сборке. Поскольку клиентский код теоретически может иметь доступ к двум таким сборкам одновременно, это может представлять серьезную проблему.
* Риск того, что данная сборка будет перезаписана другой версией той же сборки, и новая версия окажется несовместимой с некоторым существующим клиентским кодом.
* С технической точки зрения, сборка, состоящая из одного единственного модуля \*.dll или \*.ехе, называется однофайловой сборкой. В однофайловых сборках все необходимые CIL-инструкции, метаданные и манифесты содержатся в одном автономном четко определенном пакете.
* Многофайловые сборки, в свою очередь, состоят из множества файлов двоичного кода .NET, каждый из которых называется модулем (module). При построении многофайловой сборки в одном из ее модулей (называемом первичным или главным (primary) модулем) содержится манифест всей самой сборки (и, возможно, СIL-инструкции и метаданные по различным типам), а во всех остальных — манифест, CIL-инструкции и метаданные типов, охватывающие уровень только соответствующего модуля. Как нетрудно догадаться, в главном модуле содержится описание набора требуемых дополнительных модулей внутри манифеста сборки.

9. Что такое assembly manifest?

Манифест представляет собой обычный набор таблиц метаданных. Эти таблицы описывают файлы, которые входят в сборку, общедоступные экспортируемые типы, реализованные в файлах сборки, а также относящиеся к сборке файлы ресурсов или данных.

Каждая сборка, будь то статическая или динамическая, содержит набор данных, описывающих, как элементы в сборке связаны друг с другом. Манифест сборки содержит метаданные этой сборки. Манифест сборки содержит все метаданные, необходимые для указания требований к версии сборки и удостоверения безопасности, а также все метаданные, необходимые для определения области сборки и разрешения ссылок на ресурсы и классы. Манифест сборки может храниться либо в PE-файле ( .exe или .dll ) с кодом на промежуточном языке Microsoft (MSIL), либо в отдельном PE-файле, который содержит только информацию о манифесте сборки.

10.Что такое GAC?

Если сборка предназначена для совместного использования несколькими приложениями, ее нужно поместить в общеизвестный каталог, который среда CLR должна автоматически проверять при обнаружении ссылки на сборку. Место, где располагаются совместно используемые сборки, называют глобальным кэшем сборок (global assembly cache, GAC).

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).

12.Как и для чего определен метод Main?

Простейшая программа на C# состоит из одного класса, содержащего один метод. Каждая программа на C# должна иметь по меньшей мере метод *Main* (c большой буквы). **Данный метод часто называют точкой входа в программу.** Несмотря на свою важность, Main не является ключевым словом.

Также как и переменные, метод Main должен быть объявлен в программе. Кроме того, этот метод должен быть определен как *public* и *static*. Для компилятора С# не важно, в каком из классов определен метод Main. Компилятор С# достаточно «умен», чтобы самостоятельно просмотреть ваши исходники и отыскать в них метод Main. Хотя вы можете поместить метод Main в любой класс, но рекомендуется для создать отдельный класс. Метод Main может иметь входные и выходные данные. Входные данные являются параметрами метода, а выходные - возвращаемым значением. Впрочем, Main может вообще не иметь параметров и возвращаемого значения

13.Варианты использования директивы using( using Directive ) в C#.

* В начале файла исходного кода, перед любыми объявлениями пространств имен или типов.
* В любом пространстве имен, но перед любыми пространствами имен или типами, объявленными в этом пространстве имен, если не используется globalмодификатор, и в этом случае директива должна стоять перед всеми объявлениями пространств имен и типов.

14.Как связаны между собой сборки и пространства имен?

Пространства имён используют для логической группировки родственных типов, чтобы разработчику было проще найти нужный тип.

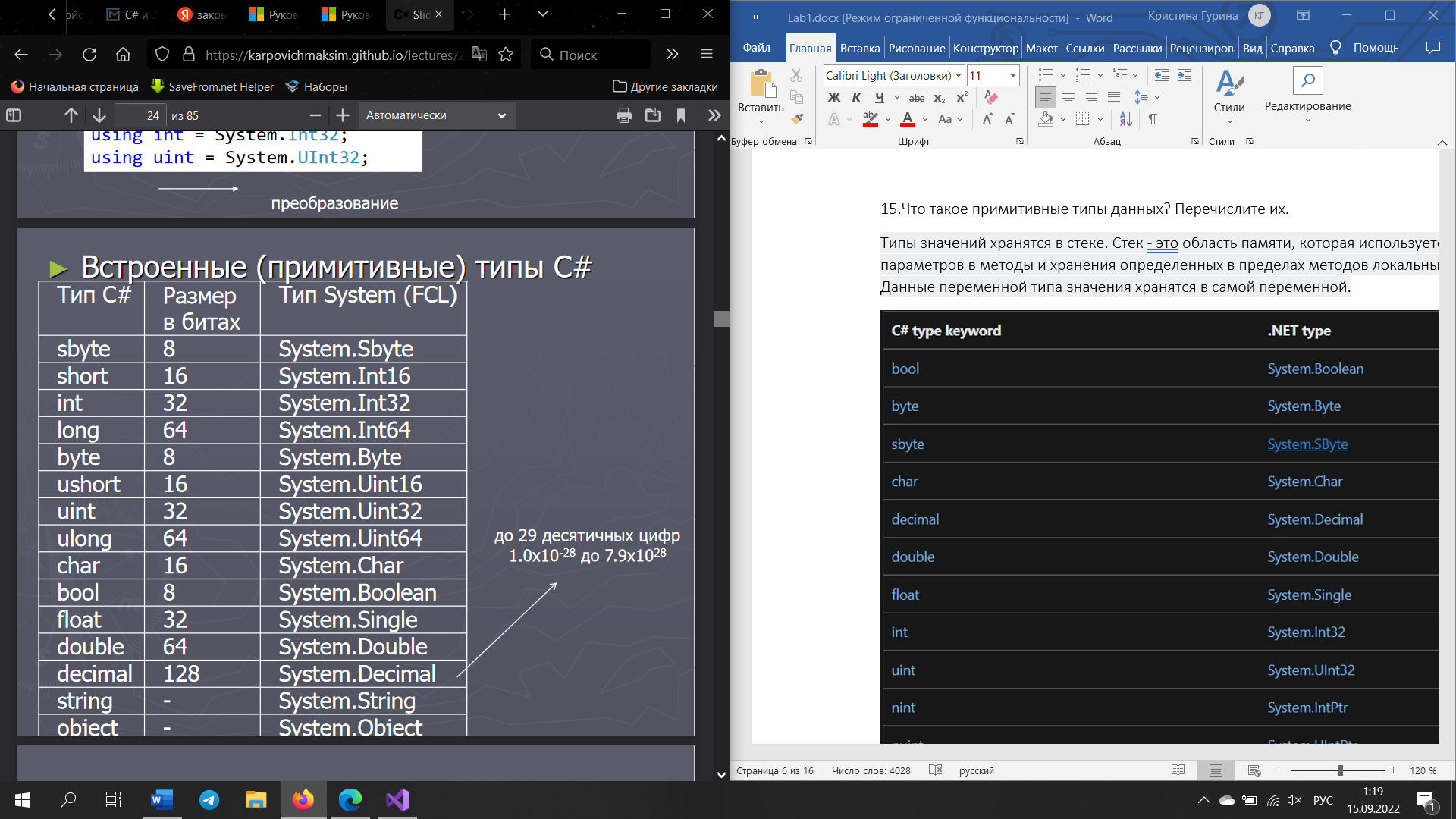
***Пространство имен*** определяет область объявлений, в которой допускается хранить одно множество имен отдельно от другого. По существу, имена, объявленные в одном пространстве имен, не будут вступать в конфликт с аналогичными именами, объявленными в другой области.

Приложения .NET создаются за счет складывания вместе некоторого количества сборок. Сборка представляет собой поддерживающий версии самоописываемый двоичный файл, обслуживаемый CLR (Common Language Runtime — общеязыковая исполняющая среда).

Пространство имен представляет собой логическую группировку типов (в основном, чтобы избежать столкновений имен). Сборка может содержать типы в нескольких пространствах имен (System.DLL содержит несколько...), и одно пространство имен может быть распространено по сборкам (например, System.Threading).

15.Что такое примитивные типы данных? Перечислите их.

Типы значений хранятся в стеке. Стек - это область памяти, которая используется для передачи параметров в методы и хранения определенных в пределах методов локальных переменных. Данные переменной типа значения хранятся в самой переменной.

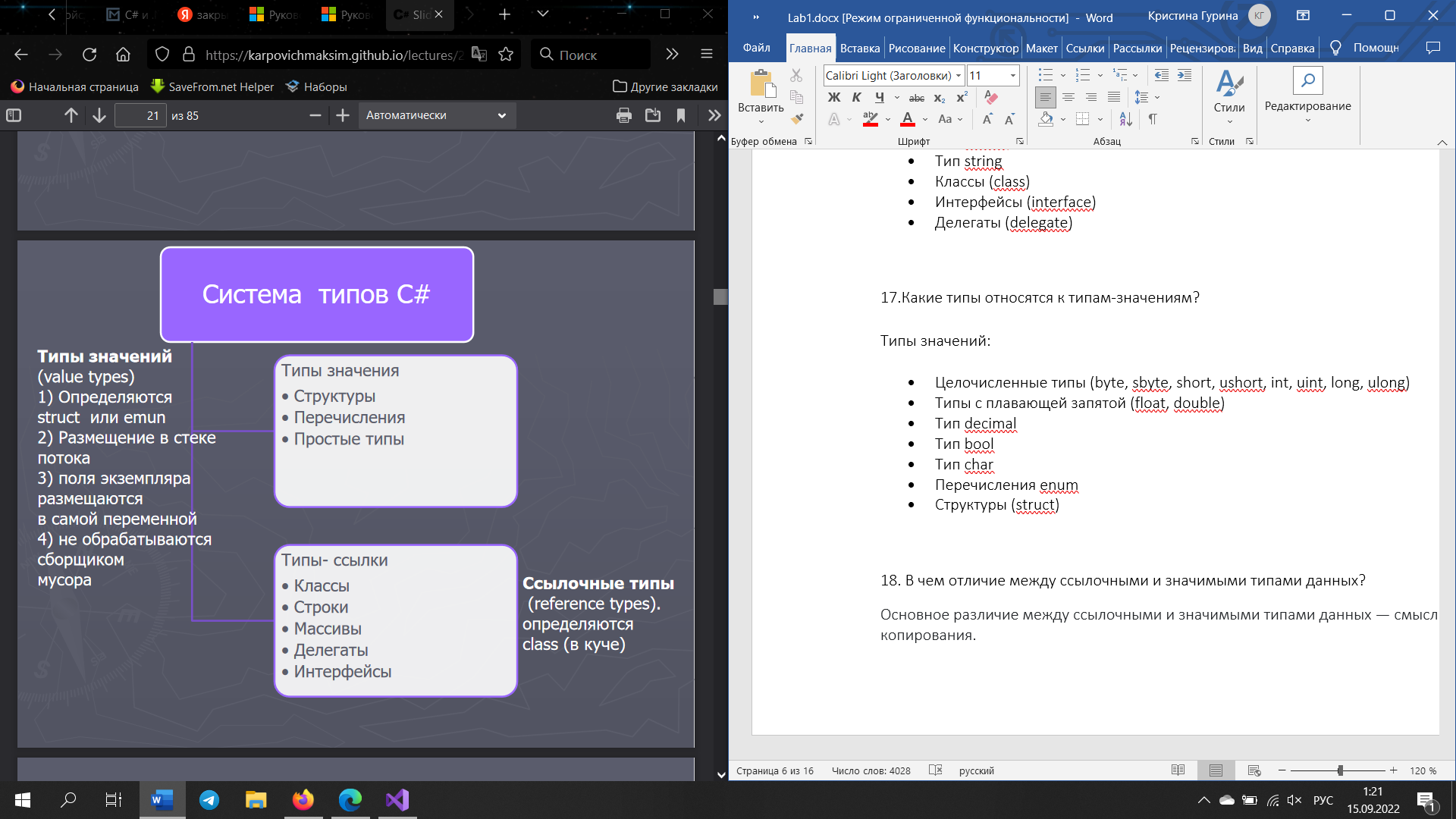


16.Что такое ссылочные типы? Какие типы относятся к ним?

Переменная ссылочного типа содержит не данные, а ссылку на них. Сами данные в этом случае уже хранятся в куче. Куча - это область памяти, в которой размещаются управляемые объекты, и работает сборщик мусора. Сборщик мусора освобождает все ресурсы и объекты, которые уже не нужны.

Ссылочные типы:

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



17.Какие типы относятся к типам-значениям?

Типы значений:

* Целочисленные типы (byte, sbyte, short, ushort, int, uint, long, ulong)
* Типы с плавающей запятой (float, double)
* Тип decimal
* Тип bool
* Тип char
* Перечисления enum
* Структуры (struct)

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

Основное различие между ссылочными и значимыми типами данных — смысл равенства и копирования.

Для значимых типов при копировании вы получаете новый экземпляр, содержащий копии значений исходного экземпляра. Иными словами, экземпляры ведут себя как значения, например, как числа. Если вы скопировали значение, и изменили исходное значение, то эти изменения никак не отразятся на копии

Для ссылочных типов, копирование даёт вам новую ссылку на те же данные. Соответственно, если вы меняете объект ссылочного типа, то эти изменения становятся доступны по любой из ссылок на объект

19.Что такое упаковка и распаковка значимых типов?

**Упаковка** это преобразование **value**(**значимого**) типа в **Object** тип

**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**

2) **Распаковка** 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.

Для каждого типа данных C# существует соответствующий тип данных в CRL (Common Language Runtime). Это, в частности, означает, что каждый тип имеет два названия – полный (из CLR, его можно использовать в любом языке .NET) и сокращенный, который используется в C#.

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?

Несмотря на то, что переменная x меняет тип своего значения несколько раз, данный код будет нормально работать. В этом использование типов dynamic отличается от применения ключевого слова **var**. Для переменной, объявленной с помощью ключевого слова var, тип выводится во время компиляции и затем во время выполнения больше не меняется.

23.Что такое неявно типизированная переменная?

начиная с версии C# 3.0, 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?

Nullable-типы являются расширением обычных типов, позволяющим хранить значение null (то есть в отличии от стандартных типов, Nullable-типы позволяют пользователю не хранить никакого значения в поле скалярного типа).

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

Строковые литералы представляют строки. Строки заключаются в двойные кавычки:

|  |  |
| --- | --- |
| 1  2  3 | Console.WriteLine("hello"); |

* **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

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?

|  |  |
| --- | --- |
| **Сравнение строк** | |
| *Compare()* | Статический метод, сравнивает строку strA со строкой strB. Возвращает положительное значение, если строка strA больше строки strB; отрицательное значение, если строка strA меньше строки strB; и нуль, если строки strA и strB равны. Сравнение выполняется с учетом регистра и культурной среды.  Если параметр ignoreCase принимает логическое значение true, то при сравнении не учитываются различия между прописным и строчным вариантами букв. В противном случае эти различия учитываются.  Параметр comparisonType определяет конкретный способ сравнения строк. Класс CultureInfo определен в пространстве имен System.Globalization. |
| Сравнивает части строк strA и strB. Сравнение начинается со строковых элементов strA[indexA] и strB[indexB] и включает количество символов, определяемых параметром length. Метод возвращает положительное значение, если часть строки strA больше части строки strB; отрицательное значение, если часть строки strA меньше части строки strB; и нуль, если сравниваемые части строк strA и strB равны. Сравнение выполняется с учетом регистра и культурной среды. |
| *CompareOrdinal()* | Делает то же, что и метод Compare(), но без учета локальных установок |
| *CompareTo()* | Сравнивает вызывающую строку со строковым представлением объекта value. Возвращает положительное значение, если вызывающая строка больше строки value; отрицательное значение, если вызывающая строка меньше строки value; и нуль, если сравниваемые строки равны |
| Сравнивает вызывающую строку со строкой strB |
| *Equals()* | Возвращает логическое значение true, если вызывающая строка содержит ту же последовательность символов, что и строковое представление объекта obj. Выполняется порядковое сравнение с учетом регистра, но без учета культурной среды |
| Возвращает логическое значение true, если вызывающая строка содержит ту же последовательность символов, что и строка value. Выполняется порядковое сравнение с учетом регистра, но без учета культурной среды. Параметр comparisonType определяет конкретный способ сравнения строк |
| Возвращает логическое значение true, если строка *a* содержит ту же последовательность символов, что и строка *b*. Выполняется порядковое сравнение с учетом регистра, но без учета культурной среды. Параметр comparisonType определяет конкретный способ сравнения строк |
| **Конкатенация (соединение) строк** | |
| *Concat()* | Комбинирует отдельные экземпляры строк в одну строку (конкатенация) |
| **Поиск в строке** | |
| *Contains()* | Метод, который позволяет определить, содержится ли в строке определенная подстрока (value) |
| *StartsWith()* | Возвращает логическое значение true, если вызывающая строка начинается с подстроки value. В противном случае возвращается логическое значение false. Параметр comparisonType определяет конкретный способ выполнения поиска |
| *EndsWith()* | Возвращает логическое значение true, если вызывающая строка оканчивается подстрокой value. В противном случае возвращает логическое значение false. Параметр comparisonType определяет конкретный способ поиска |
| *IndexOf()* | Находит первое вхождение заданной подстроки или символа в строке. Если искомый символ или подстрока не обнаружены, то возвращается значение -1 |
| Возвращает индекс первого вхождения символа или подстроки value в вызывающей строке. Поиск начинается с элемента, указываемого по индексу startIndex, и охватывает число элементов, определяемых параметром count (если указан). Метод возвращает значение -1, если искомый символ или подстрока не обнаружен |
| *LastIndexOf()* | То же, что IndexOf, но находит последнее вхождение символа или подстроки, а не первое |
| *IndexOfAny()* | Возвращает индекс первого вхождения любого символа из массива anyOf, обнаруженного в вызывающей строке. Поиск начинается с элемента, указываемого по индексу startIndex, и охватывает число элементов, определяемых параметром count (если они указаны). Метод возвращает значение -1, если не обнаружено совпадение ни с одним из символов из массива anyOf. Поиск осуществляется порядковым способом |
| *LastIndexOfAny* | Возвращает индекс последнего вхождения любого символа из массива anyOf, обнаруженного в вызывающей строке |
| **Разделение и соединение строк** | |
| *Split* | Метод, возвращающий массив string с присутствующими в данном экземпляре подстроками внутри, которые отделяются друг от друга элементами из указанного массива char или string.  В первой форме метода Split() вызывающая строка разделяется на составные части. В итоге возвращается массив, содержащий подстроки, полученные из вызывающей строки. Символы, ограничивающие эти подстроки, передаются в массиве separator. Если массив separator пуст или ссылается на пустую строку, то в качестве разделителя подстрок используется пробел. А во второй форме данного метода возвращается количество подстрок, определяемых параметром count. |
| В двух первых формах метода Split() вызывающая строка разделяется на части и возвращается массив, содержащий подстроки, полученные из вызывающей строки. Символы, разделяющие эти подстроки, передаются в массиве separator. Если массив separator пуст, то в качестве разделителя используется пробел. А в третьей и четвертой формах данного метода возвращается количество строк, ограничиваемое параметром count.  Но во всех формах параметр options обозначает конкретный способ обработки пустых строк, которые образуются в том случае, если два разделителя оказываются рядом. В перечислении StringSplitOptions определяются только два значения: *None* и *RemoveEmptyEntries*. Если параметр options принимает значение None, то пустые строки включаются в конечный результат разделения исходной строки. А если параметр options принимает значение RemoveEmptyEntries, то пустые строки исключаются из конечного результата разделения исходной строки. |
| *Join()* | Строит новую строку, комбинируя содержимое массива строк.  В первой форме метода Join() возвращается строка, состоящая из сцепляемых подстрок, передаваемых в массиве value. Во второй форме также возвращается строка, состоящая из подстрок, передаваемых в массиве value, но они сцепляются в определенном количестве count, начиная с элемента массива value[startIndex]. В обеих формах каждая последующая строка отделяется от предыдущей разделительной строкой, определяемой параметром separator. |
| **Заполнение и обрезка строк** | |
| *Trim()* | Метод, который позволяет удалять все вхождения определенного набора символов с начала и конца текущей строки.  В первой форме метода Trim() из вызывающей строки удаляются начальные и конечные пробелы. А во второй форме этого метода удаляются начальные и конечные вхождения в вызывающей строке символов из массива trimChars. В обеих формах возвращается получающаяся в итоге строка. |
| *PadLeft()* | Позволяет дополнить строку символами слева.  В первой форме метода PadLeft() вводятся пробелы с левой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth. А во второй форме данного метода символы, обозначаемые параметром paddingChar, вводятся с левой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth. В обеих формах возвращается получающаяся в итоге строка. Если значение параметра totalWidth меньше длины вызывающей строки, то возвращается копия неизмененной вызывающей строки. |
| *PadRight()* | Позволяет дополнить строку символами справа. |
| **Вставка, удаление и замена строк** | |
| *Insert()* | Используется для вставки одной строки в другую, где value обозначает строку, вставляемую в вызывающую строку по индексу startIndex. Метод возвращает получившуюся в итоге строку. |
| *Remove()* | Используется для удаления части строки. В первой форме метода Remove() удаление выполняется, начиная с места, указываемого по индексу startIndex, и продолжается до конца строки. А во второй форме данного метода из строки удаляется количество символов, определяемое параметром count, начиная с места, указываемого по индексу startIndex. |
| *Replace()* | Используется для замены части строки. В первой форме метода Replace() все вхождения символа oldChar в вызывающей строке заменяются символом newChar. А во второй форме данного метода все вхождения строки oldValue в вызывающей строке заменяются строкой newValue. |
| **Смена регистра** | |
| *ToUpper()* | Делает заглавными все буквы в вызывающей строке. |
| *ToLower()* | Делает строчными все буквы в вызывающей строке. |
| **Получение подстроки из строки** | |
| *Substring()* | В первой форме метода Substring() подстрока извлекается, начиная с места, обозначаемого параметром startIndex, и до конца вызывающей строки. А во второй форме данного метода извлекается подстрока, состоящая из количества символов, определяемых параметром length, начиная с места, обозначаемого параметром startIndex. |

28.В чем отличие пустой и null строки?

NULL - это отсутствие значения. Пустая строка - это значение, но она просто пуста.

29.Как можно выполнить сравнение строк?

27

30.В чем отличие типов String и StringBuilder?

В отличие от типа String, для типа StringBuilder отдельный символ может быть изменен по его индексу.

31.Поясните явные преобразования переменных с помощью команд Convert.

**int** d = 13;

**int** c = 4;

**var** e = (**double**) d / c;

Таким образом, переменная e получит тип double и значение 3,25. Если бы мы не использовали преобразование, то у этой переменной был бы тип int и значение 3 согласно правилам деления целых чисел в C#.

32.Как выполнить консольный ввод/вывод?

Для вывода информации на консоль мы уже использовали встроенный метод **Console.WriteLine**.

**Console.ReadLine()**. Он позволяет получить введенную строку.

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

тип\_переменной[] название\_массива;

int[] numbers;

int[] nums2 = new int[4] { 1, 2, 3, 5 };

34.Что такое ступенчатый массив? Как его задать?

Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной. Следовательно, ступенчатый массив может быть использован для составления таблицы из строк разной длины.

Ступенчатые массивы объявляются с помощью ряда квадратных скобок, в которых указывается их размерность. Например, для объявления двумерного ступенчатого массива служит следующая общая форма:

*тип [][] имя массива = new тип[размер] [];*

где размер обозначает число строк в массиве.

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

Оператор foreach выполняет оператор или блок операторов для каждого элемента в экземпляре типа

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

* Тип имеет открытый метод без параметров GetEnumerator. Начиная с C# 9.0 метод GetEnumerator может быть [методом расширения](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/extension-methods) типа.
* тип возвращаемого значения метода GetEnumerator должен содержать открытое свойство Current и открытый метод MoveNext без параметров с типом возвращаемого значения bool.
* В следующем примере показано использование оператора foreach с экземпляром типа [System.Span<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.span-1), который не реализует интерфейс:
* C#Копировать
* Span<int> numbers = new int[] { 3, 14, 15, 92, 6 };
* foreach (int number in numbers)
* {
* Console.Write($"{number} ");
* }
* // Output:
* // 3 14 15 92 6

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

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

Кортеж или Tuple — это простой способ представить набор данных, который имеет несколько связанных или не связанных друг с другом значений.

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**.

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

Так, если требуется указать, что выражение будет проверяться на переполнение, следует использовать **ключевое слово checked**, а если требуется проигнорировать переполнение — **ключевое слово unchecked**.

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

**По умолчанию операторы в C# выполняться в непроверяемом контексте**.

Примеры использования checked и unchecked:

sbyte a = 127;

// Проверять переполнение.

unchecked

{

a++; // Логическая ошибка

}

// 127 + 1 = -128

Console.WriteLine(a);

sbyte b = 127;

// Проверять переполнение.

checked

{

b++; // ОШИБКА уровня компилятора

}

// 127 + 1 = -128

Console.WriteLine(b);

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

Сборщик мусора C# может переместить объекты в памяти в соответствии с алгоритмом процесса уборки мусора. Язык C# предоставляет специальное ключевое слово **fixed**, чтобы указать сборщику мусора не перемещать объект. Это означает, что позиция переменной в памяти в памяти фиксируется, чтобы на нее мог ссылаться указатель. На C# это называется прикреплением (pinning).