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

Microsoft.NET (.NET Framework) – программная платформа.( обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows) Содержит следующие основные компоненты:

CLR (Common Language Runtime) – общеязыковая среда исполнения, виртуальная машина на которой исполняются все приложения, работающие в среде .NET. Реализация CLI VES компанией Microsoft. Компилятор JIT(Just in Time).

MSIL(Microsoft IL) – реализация CLI CIL компанией Microsoft.

FCL (Framework Class Library) – реализация CLI BCL компанией Microsoft. Можно рассматривать, как API CRL.

**2. Поясните, что такое CLR-среда.**

CLR (Common Language Runtime) – Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация)

**3. Что такое FCL?**

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

**4. Какая наименьшая исполнимая единица в .NET?**

Сборка

**5. Что такое IL?**

IL объектно-ориентированный машинный язык не зависящий от процессора

ILAsm exe – ассемблер

ILDasm exe - дизассемблер IL

**6. Пояснить работу JIT-компилятора?**

JIT-компилятор (Just-In-Time)

1) CLR ищет типы данных и загружает во внутренние структуры

2) Для каждого метода CLR заносит адрес внутренней CLR функции JITCompiler

3) JITCompiler ищет в метаданных соответствующей сборки IL-код вызываемого метода, проверяет и компилирует IL-код в машинные команды

4) Они хранятся в динамически выделенном блоке памяти.

5) JITCompiler заменяет адрес вызываемого метода адресом блока памяти, содержащего готовые машинные команды

6) JITCompiler передает управление коду в этом блоке памяти.

**7. Что такое CTS (Common Type System)?**

CTS (Common Type Systems)- спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Microsoft выпустил несколько компиляторов соответствующих этой спецификации: С++/CLI (C++ с управляемыми расширениями), С#, VB .NET, JScript.

**8. Какие аспекты поведения определяет тип System.Object?**

**9. Что находится в MSCorLib dll?**

**10. Что такое частные и общие сборки?**

Частные находятся в каталоге программы, а общие в GAC. Global Assembly Cache (англ. GAC — Глобальный Кэш Сборок) — в инфраструктуре .NET — подсистема, хранящая сборки CLI

Когда мы создаем приложение в результате компиляции в Visual Studio или в консоли, результатом этой работы является файл exe или dll (в зависимости от выбранных настроек), который называется сборкой приложения. Сборка является базовой структурной единицей в .NET, на уровне которой проходит контроль версий, развертывание и конфигурация приложения.

**11. Что такое assembly manifest?**

Ключевым компонентом сборки является ее манифест. Если у сборки отсутствует манифест, то заключенный в ней код MSIL выполняться не будет. Манифест может находиться в одном файле с исполняемым кодом сборки, а может размещаться и в отдельном файле.

Манифест хранит следующие данные:

Имя сборки

Номер версии: основной и дополнительный номера. Используется для управления версиями

Язык и региональные параметры: информация о языке и региональных параметрах, которые поддерживает сборка

Информация о строгом имени: открытый ключ издателя

Список всех файлов сборки: хэш и имя каждого из входящих в сборку файлов

Список ссылок на другие сборки, которые использует текущая сборка

Список ссылок на типы, используемые сборкой

Таким образом, манифест позволяет системе определить все файлы, входящие в сборку, сопоставить ссылки на типы, ресурсы, сборки с их файлами, управлять контролем версий.

**12. Что такое GAC?**

Global Assembly Cache (англ. GAC — Глобальный Кэш Сборок) — в инфраструктуре .NET — подсистема, хранящая сборки CLI

**13. Чем managed code отличается от unmanaged code**

Это означает, что CLR заботится об управлении памятью, о совместимости между платформами, о безопасности кода и так далее Это хорошо описано у Рихтера.

Управляемый код - это код .NET(VB.NET, С# и т.д.), который вы пишете и компилируете в .NET [CIL](http://en.wikipedia.org/wiki/Common_Intermediate_Language).

Неуправляемый код - это код, который не находится под .NET, который компилируется для прямого машинного кода.

<http://qaru.site/questions/63811/what-is-the-difference-in-managed-and-unmanaged-code-memory-and-size>

**14. 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.

Основной точкой входа в программу является функция Main()

**15. Варианты использования директивы using( using Directive ) в C#.**

Директива using разрешает использование типов в пространстве имен, поэтому уточнение использования типа в этом пространстве имен не требуется:

using System.Text;

Также директива using позволяет создавать псевдонимы пространства имен или типа:

using Project = PC.MyCompany.Project;

Эта версия использования называется директива using alias.

**16. Как связаны между собой сборки и пространства имен?**

Сборки являются структурными элементами приложений .NET Framework; они составляют основную единицу развертывания, управления версиями, повторного использования, областей действия активации и разрешений безопасности. Сборка представляет собой коллекцию типов и ресурсов, собранных для совместной работы и образующих логическую функциональную единицу. Сборка предоставляет общеязыковой исполняющей среде сведения, необходимые для распознавания реализаций типов. Для среды выполнения тип не существует вне контекста сборки.  
Никакого упоминания пространств имен, только типы (классы, структуры и т.д.).  
Пространство имен Цитата с MSDN (полный текст):  
В программировании на C# пространства имен используются с полной нагрузкой по двум направлениям. Во-первых, платформа .NET Framework использует пространства имен для организации множества классов. Во-вторых, объявление собственного пространства имен поможет в управлении областью действия имен классов и методов в крупных программных проектах.  
Опять же никакого упоминания сборок.  
тогда я не знаю, такой вопрос в лабораторной  
Скорее всего от вас хотят услышать, что перед использованием using SomeNameSpace;, вы должны указать в проекте ссылку на сборку, в которой объявлены типы данного пространства имен. Но это не ни как не влияет на то, что сборки и пространства имен не имеют жесткой связи на уровне языка и платформы. Пространства имен - всего лишь агрегатная приставка к имени типа для явного указания принадлежности типа к определенной группе логически связанных типов, не более того. А вот типы, в отличие от пространств имен, имеют жесткую связь со сборкой в которой находятся, об этом можно подробнее почитать в первой ссылке.

**17. Что такое примитивные типы данных? Перечислите их.**

В языке C# есть следующие примитивные типы данных:

* **bool**: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean

|  |  |
| --- | --- |
| 1  2 | bool alive = true;  bool isDead = false; |

* **byte**: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte

|  |  |
| --- | --- |
| 1  2 | byte bit1 = 1;  byte bit2 = 102; |

* **sbyte**: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte

|  |  |
| --- | --- |
| 1  2 | sbyte bit1 = -101;  sbyte bit2 = 102; |

* **short**: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16

|  |  |
| --- | --- |
| 1  2 | short n1 = 1;  short n2 = 102; |

* **ushort**: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16

|  |  |
| --- | --- |
| 1  2 | ushort n1 = 1;  ushort n2 = 102; |

* **int**: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

|  |  |
| --- | --- |
| 1  2  3 | int a = 10;  int b = 0b101;  // бинарная форма b =5  int c = 0xFF;   // шестнадцатеричная форма c = 255 |

* **uint**: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32

|  |  |
| --- | --- |
| 1  2  3 | uint a = 10;  uint b = 0b101;  uint c = 0xFF; |

* **long**: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64

|  |  |
| --- | --- |
| 1  2  3 | long a = -10;  long b = 0b101;  long c = 0xFF; |

* **ulong**: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64

|  |  |
| --- | --- |
| 1  2  3 | ulong a = 10;  ulong b = 0b101;  ulong c = 0xFF; |

* **float**: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта. Представлен системным типом System.Single
* **double**: хранит число с плавающей точкой от ±5.0\*10-324 до ±1.7\*10308 и занимает 8 байта. Представлен системным типом System.Double
* **decimal**: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0\*10-28 до ±7.9228\*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal
* **char**: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

|  |  |
| --- | --- |
| 1  2  3 | char a = 'A';  char b = '\x5A';  char c = '\u0420'; |

* **string**: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют символьные литералы.

|  |  |
| --- | --- |
| 1  2 | string hello = "Hello";  string word = "world"; |

* **object**: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

|  |  |
| --- | --- |
| 1  2  3 | object a = 22;  object b = 3.14;  object c = "hello code"; |

**18. Что такое ссылочные типы? Какие типы относятся к ним?**

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/keywords/delegate)

В C# также предусмотрены следующие встроенные ссылочные типы:

* [dynamic](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/dynamic)
* [object](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/object)
* [string](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/string)

**19. 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)

**20. В чем отличие между ссылочными и значимыми типами данных?**

Значимые типы хранят значение, а ссылочные - ссылку на значение.

**21. Что такое упаковка и распаковка значимых типов?**

* Когда любой значимый тип присваивается к ссылочному типу данных, значение перемещается из области стека в кучу. Эта операция называется упаковкой.
* Когда любой ссылочный тип присваивается к значимому типу данных, значение перемещается из области кучи в стек. Это называется распаковкой.

**22. Для чего используется тип dynamic?**

dynamic можно рассматривать как специализированную форму System.Object, в том смысле, что типу данных dynamic может быть присвоено любое значение.

**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?**

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

Строковые литералы представляют строки. Строки заключаются в двойные кавычки. Если внутри строки необходимо вывести двойную кавычку, то такая внутренняя кавычка предваряется обратным слешем.

СРАВНЕНИЕ, ОБЪЯВЛЕНИЕ, ИНИЦИАЛИЗАЦИЯ,

**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?**

В классе System.String предоставляется набор методов для определения длины символьных данных, поиска подстроки в текущей строке, преобразования символов из верхнего регистра в нижний и наоборот, и т.д.

**28. В чем отличие пустой и null строки?**

Используйте null, если вы хотите представить, что нет значения;

Используйте String.Empty, когда вы хотите представить, что есть значение, но значение представляет собой пустую строку.

**29. Как можно выполнить сравнение строк?**

|  |  |
| --- | --- |
| [String.Compare](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compare) | Сравнивает значения двух строк. Возвращает целочисленное значение. |
| [String.CompareOrdinal](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compareordinal) | Сравнивает две строки без учета локального языка и региональных параметров. Возвращает целочисленное значение. |
| [String.CompareTo](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.compareto) | Сравнивает текущий строковый объект с другой строкой. Возвращает целочисленное значение. |
| [String.StartsWith](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.startswith) | Определяет, начинается ли строка с переданной строки. Возвращает логическое значение. |
| [String.EndsWith](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.endswith) | Определяет, заканчивается ли строка переданной строкой. Возвращает логическое значение. |
| [String.Equals](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.equals) | Определяет, совпадают ли две строки. Возвращает логическое значение. |
| [String.IndexOf](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.indexof) | Возвращает индекс позиции символа или строки начиная с начала проверяемой строки. Возвращает целочисленное значение. |
| [String.LastIndexOf](https://docs.microsoft.com/ru-ru/dotnet/api/system.string.lastindexof) | Возвращает индекс позиции символа или строки начиная с конца проверяемой строки. Возвращает целочисленное значение. |

**30. В чем отличие типов String и StringBuilder?**

string экземпляр является неизменным. Вы не можете изменить его после его создания. Любая операция, которая появляется для изменения строки, возвращает новый экземпляр

Если вам нужна изменчивая строка, например, которую вы производите по кусочкам или где вы меняете много вещей, вам понадобится StringBuilder, который является буфером символов, который можно изменить.

**31. Поясните явные преобразования переменных с помощью команд Convert.**

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

short s = 45;

    byte b = Convert.ToByte(s);

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

    long l = 1234;

    int i = Convert.ToInt32(l);

**32. Как выполнить консольный ввод/вывод?**

Console.WriteLine("Вы ввели число {0}", i);

    Console.ReadLine();

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

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

foreach (int i in numbers)

{

    Console.WriteLine(i);

}

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

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

int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };

foreach (int i in mas)

    Console.Write($"{i} ");

Console.WriteLine();

**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.

int i = 0;

// Объявляем ступенчатый массив

int[][] myArr = new int[4][];

myArr[0] = new int[4];

myArr[1] = new int[6];

myArr[2] = new int[3];

myArr[3] = new int[4];

// Инициализируем ступенчатый массив

for (; i < 4; i++)

{

myArr[0][i] = i;

Console.Write("{0}\t",myArr[0][i]);

}

Console.WriteLine();

for (i = 0; i < 6; i++)

{

myArr[1][i] = i;

Console.Write("{0}\t", myArr[1][i]);

}

Console.WriteLine();

for (i = 0; i < 3; i++)

{

myArr[2][i] = i;

Console.Write("{0}\t", myArr[2][i]);

}

Console.WriteLine();

for (i = 0; i < 4; i++)

{

myArr[3][i] = i;

Console.Write("{0}\t", myArr[3][i]);

}

Console.ReadLine();

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

Он может применяться к экземпляру любого типа, который удовлетворяет следующим условиям:

* включает открытый метод GetEnumerator без параметров со следующим типом возвращаемого значения: класс, структура или тип интерфейса;
* тип возвращаемого значения метода GetEnumerator должен содержать открытое свойство Current и открытый метод MoveNext без параметров с типом возвращаемого значения [Boolean](https://docs.microsoft.com/ru-ru/dotnet/api/system.boolean).

int

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

Кортеж представляет набор значений, заключенных в круглые скобки:

|  |  |
| --- | --- |
|  | var tuple = (5, 10);  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 также кортеж может передаваться в качестве параметра в метод:

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

**37. Что такое локальная функция?**

Локальные функции представляют функции, определенные внутри других методов.