**Базовый уровень: теория.**

1. **Изучите основные типы .NET. В чем заключается разница между int и System.Int32? double и System.Double и т.д.?**

Ключевые слова для встроенных типов C#, которые являются псевдонимами предопределенных типов в пространстве имен System.

Типы данных, которые поддерживаются компилятором напрямую, называются примитивными (primitivetypes) или встроенными у них существуют прямые аналоги в библиотеке классов .NET FrameworkClassLibrary

**2. Ознакомьтесь с var и dynamic**:

Ключевым моментом использования DLR в C# является применение типов **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.

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.

**5. Ознакомьтесь с возможностями работы со строками:**

|  |  |
| --- | --- |
| **Compare** | Сравнение двух строк в алфавитном порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т.д. |
| **CompareOrdinal** | Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки |
| **CompareTo** | Сравнение текущего экземпляра строки с другой строкой |
| **Concat** | Конкатенация строк. Метод допускает сцепление произвольного числа строк |
| **Copy** | Создание копии строки |
| **Format** | Форматирование в соответствии с заданными спецификаторами формата |
| **IndexOf, LastIndexOf,…** | Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора |
| **Insert** | Вставка подстроки в заданную позицию |
| **Join** | Слияние массива строк в единую строку. Между элементами массива вставляются разделители (см. далее) |
| **Length** | Длина строки (количество символов) |
| **Remove** | Удаление подстроки из заданной позиции |
| **Replace** | Замена всех вхождений заданной подстроки или символа новой подстрокой или символом |
| **Split** | Разделение строки на элементы, используя заданные разделители. Результаты помещаются в массив строк |
| **Substring** | Выделение подстроки, начиная с заданной позиции |

Хотя класс System.String предоставляет нам широкую функциональность по работе со строками, все-таки он имеет свои недостатки. Прежде всего, объект String представляет собой неизменяемую строку. Когда мы выполняем какой-нибудь метод класса String, система создает новый объект в памяти с выделением ему достаточного места.

Чтобы выйти из этой ситуации во фреймворк .NET был добавлен новый класс **StringBuilder**, который находится в пространстве имен System.Text. Этот класс представляет динамическую строку.

Microsoft рекомендует использовать класс String в следующих случаях:

* При небольшом количестве операций и изменений над строками
* При выполнении фиксированного количества операций объединения. В этом случае компилятор может объединить все операции объединения в одну
* Когда надо выполнять масштабные операции поиска при построении строки, например, IndexOf или StartsWith. Класс StringBuilder не имеет подобных методов.

Класс StringBuilder рекомендуется использовать в следующих случаях:

* При неизвестном количестве операций и изменений над строками во время выполнения программы
* Когда предполагается, что приложению придется сделать множество подобных операций

**6. Изучите возможности форматирования строк:**

**Вопросы:**

1. **Что такое .NET Framework? Из чего он состоит?**

Обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.

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

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

***MSIL (Microsoft IL)***–реализация CLICIL компанией Microsoft.

***FCL (Framework Class Library)***–реализация CLIBCL компанией Microsoft. Можно рассматривать, как APICRL.

1. **Что такое CLR, FCL/BCL, CLI, IL?**

**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)

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

**IL** – язык ассемблера.

1. **Как работает JIT-компилятор?**

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 JIT -Compiler

3) JIT-Compiler 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) JIT-Compiler заменяет адрес вызываемого метода адресом блока памяти, содержащего готовые машинные команды

6) JIT-Compiler передает управление коду в этом блоке памяти.

1. **Что такое CTS?**

**CTS (Common Type Systems)** - спецификация 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.

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

Наименьшая исполняемая единица .NET. Состоит из манифеста и управляемого модуля.

- это абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов.

1. **Какие виды сборок существуют?**

Глобальная и локальная.

1. **Что такое GAC?**

GAC – Global Assembly Cache

1. **Как происходит поиск нужной сборки?**

CLR.

1. **Что содержит mscorlib.dll?**

**10. Какая наименьшая исполняемая единица .NET?**

Сборка.

1. **Для чего служит метод Main? Какова его сигнатура? Для чего служат его параметры?**

Меин – точка входа в программу.

Сигнатура складывается из следующих аспектов:

* Имя метода
* Количество параметров
* Типы параметров
* Порядок параметров
* Модификаторы параметров

Для компиляции через консоль – Андрей.

**12. Для чего служит директива using? Какие существуют варианты ее использования?**

Директива **using** используется в следующих трех целях.

* Для разрешения использования типов в пространстве имен, чтобы не нужно было квалифицировать использование типа в этом пространстве имен:

using System.Text;

* Для разрешения доступа к статическим членам и вложенным типам без необходимости квалифицировать доступ с помощью имени типа.

using static System.Math;

* Чтобы создать псевдоним для пространства имен или типа. Это называется *директивой using static*.

using Project = PC.MyCompany.Project;

Ключевое слово using также используется для создания *операторов using*, которые помогают обеспечить правильную обработку объектов [IDisposable](https://docs.microsoft.com/ru-ru/dotnet/api/system.idisposable), таких как файлы и шрифты. Дополнительные сведения см. в разделе [Оператор using](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/using-statement).

**13. Какие типы данных существуют в .NET?**

Ссылочные и значимые.

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

Типы данных, которые поддерживаются компилятором напрямую, называются примитивными (primitivetypes) или встроенными у них существуют прямые аналоги в библиотеке классов .NET Framework Class Library

**15. В чем разница между ссылочными типами данных и значимыми?**

**Ссылочные типы (**reference types) - определяются class (в куче).

**Типы значений** (value types)

1) Определяются struct или enum

2) Размещение в стеке потока

3) поля экземпляра размещаются в самой переменной

4) не обрабатываются сборщиком мусора

|  |  |  |
| --- | --- | --- |
| размещение | В стеке потока | В управляемой куче |
| формы | В неупакованной (unboxed) и упакованной (boxed) | В упакованной (boxed) |
| Наследование | System.ValueType  (есть те же методы) | System.Object |
| по умолчанию присваивается | 0 | null  может привести к  NullReferenceException |
| Операция = | выполняется копирование всех полей | копируется только адрес |
| Освобождение | нет | Требует уборки мусора |
| Освобождение памяти | Сразу | Ожидает уборки мусора |
| Не преднамеренное изменение | Имеет собственную копию данных (не возможно) | Могут ссылаться на один объект в куче (можно) |

**16. Какие примитивные типы относятся к ссылочным? К значимым?**

Ссылочные типы:

* Тип object
* Тип string

**17. В чем основная разница между string и StringBuilder?**

Хотя класс System.String предоставляет нам широкую функциональность по работе со строками, все-таки он имеет свои недостатки. Прежде всего, объект String представляет собой неизменяемую строку. Когда мы выполняем какой-нибудь метод класса String, система создает новый объект в памяти с выделением ему достаточного места.

Чтобы выйти из этой ситуации во фреймворк .NET был добавлен новый класс **StringBuilder**, который находится в пространстве имен System.Text. Этот класс представляет динамическую строку.

Microsoft рекомендует использовать класс String в следующих случаях:

* При небольшом количестве операций и изменений над строками
* При выполнении фиксированного количества операций объединения. В этом случае компилятор может объединить все операции объединения в одну
* Когда надо выполнять масштабные операции поиска при построении строки, например, IndexOf или StartsWith. Класс StringBuilder не имеет подобных методов.

Класс StringBuilder рекомендуется использовать в следующих случаях:

* При неизвестном количестве операций и изменений над строками во время выполнения программы
* Когда предполагается, что приложению придется сделать множество подобных операций

**18. В чем заключается главное отличие между var и dynamic?**

Ключевым моментом использования DLR в C# является применение типов **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.

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.

**19. Что такое упаковка и распаковка типов?**

Упаковкой (boxing) называется процесс преобразования типа значения в тип System.Object или в тип интерфейса, который реализуется данным типом-значением

1. в управляемой куче выделяется память

2. поля копируются

3. возвращается адрес объекта

Распаковка (unboxing) - получение указателя на исходный значимый тип (поля данных), содержащийся в объекте.

Объекты значимого типа существуют в двух формах: неупакованной (unboxed) и упакованной (boxed).

Ссылочные типы бывают только в упакованной форме.

**20. Для чего используется тип Nullable?**

Значение null по умолчанию могут принимать только объекты ссылочных типов. Однако в различных ситуациях бывает удобно, чтобы объекты числовых типов данных имели значение null, то есть были бы не определены. Для этого надо использовать знак вопроса ? после типа значений.

**21. Как можно преобразовать один тип в другой? Перечислите все возможные способы.**

Явное, неявное преобразование, ConvertTo, Parse.

**22. Какие виды массивов существуют в .NET?**

Одномерные, многомерные, ступенчатые.

**23. Что такое локальная функция? Какова область ее видимости?**

Начиная с версии 7.0 в языке C# поддерживаются локальные функции. Локальные функции представляют собой частные методы типа, вложенные в другой элемент. Они могут вызываться только из того элемента, в который вложены.

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

Строка кода C# может выполняться как в проверенном, так и в непроверенном контексте. В проверенном контексте арифметическое переполнение создает исключение. В непроверяемом контексте арифметическое переполнение игнорируется и результат усекается путем удаления старших разрядов, которые не помещаются в целевой тип данных.

* [checked](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/checked). Укажите проверенный контекст.
* [unchecked](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/unchecked). Укажите непроверенный контекст.

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

Если ни checked, ни unchecked не указаны, контекст по умолчанию для неконстантных выражений (выражения, вычисляемые во время выполнения) определяется по значению параметра компилятора [-checked](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/compiler-options/checked-compiler-option). По умолчанию значение этого параметра не задано, и арифметические операции выполняются в непроверенном контексте.

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

Оператор fixed не позволяет сборщику мусора переносить перемещаемую переменную. Оператор fixed допускается только в [небезопасном](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/unsafe) контексте. Можно также использовать ключевое слово fixed для создания [буферов фиксированного размера](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/unsafe-code-pointers/fixed-size-buffers).

Оператор fixed задает указатель на управляемую переменную и "закрепляет" эту переменную во время выполнения оператора. Указатели на перемещаемые управляемые переменные полезны только в контексте fixed. Без контекста fixed при сборке мусора эти переменные могут переноситься непредсказуемым образом. Компилятор C# позволяет присвоить указатель только управляемой переменной в операторе fixed.

**27. Объясните использование интерфейса *IDisposable*.**

Предоставляет механизм для освобождения неуправляемых ресурсов.

В основном этот интерфейс используется для высвобождения неуправляемых ресурсов. Сборщик мусора автоматически освобождает память, выделенную управляемому объекту, если этот объект больше не используется. Однако невозможно предсказать, когда произойдет сборка мусора. Более того, сборщик мусора не имеет сведений о неуправляемых ресурсах, таких как дескрипторы окон, или открытых файлах и потоках.

**28. Объясните работу конструкции using.**

**29. Поясните (вплоть до поколений) работу сборщика мусора в .NET.**

Ранее в теме [Типы значений и ссылочные типы](https://metanit.com/sharp/tutorial/2.16.php) мы рассматривали отдельные типы данных и как они располагаются в памяти. Так, при использовании переменных типов значений в методе, все значения этих переменных попадают в стек. После завершения работы метода стек очищается.

При использовании же ссылочных типов, например, объектов классов, для них также будет отводиться место в стеке, только там будет храниться не значение, а адрес на участок памяти в хипе или куче, в котором и будут находиться сами значения данного объекта. И если объект класса перестает использоваться, то при очистке стека ссылка на участок памяти также очищается, однако это не приводит к немедленной очистке самого участка памяти в куче. Впоследствии сборщик мусора (garbage collector) увидит, что на данный участок памяти больше нет ссылок, и очистит его.

Сборщик мусора не запускается сразу после удаления из стека ссылки на объект, размещенный в куче. Он запускается в то время, когда среда CLR обнаружит в этом потребность, например, когда программе требуется дополнительная память.

Как правило, объекты в куче располагаются неупорядочено, между ними могут иметься пустоты. Куча довольно сильно фрагментирована. Поэтому после очистки памяти в результате очередной сборки мусора оставшиеся объекты перемещаются в один непрерывный блок памяти. Вместе с этим происходит обновление ссылок, чтобы они правильно указывали на новые адреса объектов.

**30. В чем разница между *short weak reference* и *long weak reference*?**