# **­­­­№ 1 Основы CLR и .NET. Типы. Массивы, кортежи и строки**

## Задание

1. **Типы**
   1. Определите переменные всех возможных примитивных типов С# и проинициализируйте их. **Осуществите ввод и вывод их значений используя консоль.** ( <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/built-in-types>, <https://docs.microsoft.com/en-us/dotnet/api/system.console?view=netframework-4.8>)
   2. Выполните 5 операций явного и 5 неявного приведения. ([https://docs.microsoft.com/en-us/dotnet/csharp/languagereference/language-specification/conversions#implicit-conversions](https://docs.microsoft.com/en-us/dotnet/csharp/languagereference/language-specification/conversions" \l "implicit-conversions) ) **Изучите возможности класса *Convert*.**
   3. Выполните упаковку и распаковку **значимых** типов.
   4. Продемонстрируйте работу с неявно типизированной переменной.
   5. Продемонстрируйте пример работы с ***Nullable*** переменной (<https://docs.microsoft.com/en-us/dotnet/api/system.nullable-1?view=netframework-4.8>

https://docs.microsoft.com/en-us/dotnet/api/system.nullable-1?view=netcore-3.1).

* 1. . Определите переменную типа *var* и присвойте ей любое значение. Затем следующей инструкцией присвойте ей значение другого типа. Объясните причину ошибки.

1. **Строки**
   1. Объявите строковые литералы. Сравните их.
   2. Создайте три строки на основе *String*. Выполните: сцепление, копирование, выделение подстроки, разделение строки на слова, вставки подстроки в заданную позицию, удаление заданной подстроки. (<https://docs.microsoft.com/ru-ru/dotnet/api/system.string?view=netcore-3.1> ) Продемонстрируйте интерполирование строк.
   3. Создайте пустую и *null* строку. Продемонстрируйте использование метода *string.IsNullOrEmpty*. Продемонстрируйте что еще можно выполнить с такими строками
   4. Создайте строку на основе *StringBuilder*. Удалите определенные позиции и добавьте новые символы в начало и конец строки. (<https://docs.microsoft.com/ru-ru/dotnet/api/system.text.stringbuilder?view=netcore-3.1> )
2. **Массивы (** <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/> **)**
   1. Создайте целый двумерный массив и выведите его на консоль в отформатированном виде (матрица).
   2. Создайте **одномерный** массив строк. Выведите на консоль его содержимое, длину массива. Поменяйте произвольный элемент (пользователь определяет позицию и значение).
   3. Создайте **ступечатый** (не выровненный) массив вещественных чисел с 3-мя строками, в каждой из которых 2, 3 и 4 столбцов соответственно. Значения массива введите с консоли.
   4. Создайте **неявно типизированные переменные** для хранения массива и строки.
3. **Кортежи (** <https://docs.microsoft.com/en-us/dotnet/csharp/tuples> **)**
   1. Задайте кортеж из 5 элементов с типами *int, string, char, string, ulong*.
   2. Выведите кортеж на консоль целиком и выборочно ( например 1, 3, 4 элементы)
   3. Выполните распаковку кортежа в переменные. Продемонстрируйте различные способы распаковки кортежа. Продемонстрируйте использование переменной ( \_ ). (доступно начиная с C#7.3)
   4. Сравните два кортежа.
4. Создайте **локальную функцию** в main и вызовите ее (<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/local-functions> ) . Формальные параметры функции – массив целых и строка. Функция должна вернуть кортеж, содержащий: максимальный и минимальный элементы массива, сумму элементов массива и первую букву строки .
5. Работа с *checked*/*unchecked*: ( <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/checked-and-unchecked> )
   1. Определите две **локальные** функции.
   2. Разместите в одной из них блок *checked*, в котором определите переменную типа *int* с максимальным возможным значением этого типа. Во второй функции определите блок *unchecked* с таким же содержимым.
   3. Вызовите две функции. Проанализируйте результат.
6. Загрузите проект в свой репозиторий на GitHub.
7. Подготовить ответы на все вопросы используя

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types>

## Дополнительно

1. Ознакомьтесь с концепцией «небезопасного кода и указателей» в .NET. Познакомьтесь с ключевыми словами unsafe и fixed. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/unsafe-code>
2. Ознакомьтесь с работой сборщика мусора (garbage collector, GC) в .NET. <https://docs.microsoft.com/en-us/dotnet/standard/garbagecollection/fundamentals>

<http://sergeyteplyakov.blogspot.com/2012/10/net.html> <https://habr.com/ru/post/463213/>

1. Ознакомьтесь с конструкцией using

## <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-statement>

## Вопросы

* 1. Что такое .Net Framework и из чего он состоит?
  2. Что такое CLR, FCL/BCL, CLI, IL?
  3. Пояснить работу JIT-компилятора?
  4. Что такое CTS (Common Type System)?
  5. Какие аспекты поведения определяет тип System.Object?
  6. Что находится в mscorlib dll?
  7. Что такое «сборка»? Из чего состоит сборка .NET?
  8. Какие виды сборок существуют?
  9. Что такое assembly manifest?
  10. Что такое GAC?
  11. Чем managed code отличается от unmanaged code
  12. **Как и для чего определен метод Main?**
  13. **Варианты использования директивы using**( using Directive ) в C#.
  14. Как связаны между собой сборки и пространства имен?
  15. **Что такое примитивные типы данных?** Перечислите их.
  16. **Что такое ссылочные типы? Какие типы относятся к ним?**
  17. **Какие типы относятся к типам-значениям?**
  18. **В чем отличие между ссылочными и значимыми типами данных?**
  19. **Что такое упаковка и распаковка значимых типов?**
  20. В чем заключается разница между int и System.Int32? double и System.Double и т.д.?
  21. Для чего используется тип dynamic?
  22. **В чем заключается главное отличие между var и dynamic?**
  23. **Что такое неявно типизированная переменная?**
  24. **Для чего используют Nullable тип?**
  25. Как объявить строковый литерал? Какие операции можно выполнять со строкой?
  26. Какие есть способы для задания и инициализации строк?
  27. Какие методы есть у типа String?
  28. В чем отличие пустой и null строки?
  29. **Как можно выполнить сравнение строк?**
  30. **В чем отличие типов String и StringBuilder?**
  31. Поясните явные преобразования переменных с помощью команд Convert.
  32. Как выполнить консольный ввод/вывод?
  33. Приведите примеры определения и инициализации одномерных и двумерных массивов.
  34. **Что такое ступенчатый массив?** Как его задать?
  35. **Какие типы можно использовать в foreach**? Приведите пример.
  36. Что такое кортеж? Для чего и как он используется?
  37. **Что такое локальная функция?** Какова область ее видимости?
  38. В чем разница между кодом, заключенным в блок checked и кодом, заключенным в блок unchecked?
  39. Какой контекст (checked/unchecked) применяется по умолчанию? Как можно переопределить это поведение?
  40. Для чего используется ключевое слово fixed? Каковы особенности его использования?

## 1. \*\*Что такое .NET Framework и из чего он состоит?\*\*

## .NET Framework — это платформа для создания и выполнения приложений, разработанная Microsoft. Она упрощает разработку, предоставляя готовые инструменты для работы с памятью, потоками, файлами, сетями и т.д. Основные компоненты:

## - \*\*CLR (Common Language Runtime)\*\* — это движок, который управляет выполнением программ. Он отвечает за такие функции, как сборка мусора (освобождение памяти), управление безопасностью, управление потоками и многое другое.

## - \*\*FCL (Framework Class Library)\*\* — библиотека, включающая классы и методы для решения типичных задач (например, работа с файлами, сетями, графикой).

## 2. \*\*Что такое CLR, FCL/BCL, CLI, IL?\*\*

## - \*\*CLR\*\* — это "сердце" .NET, которое запускает приложения, написанные на разных языках программирования. Оно следит за тем, чтобы программа правильно использовала память, защищала данные и корректно завершалась.

## - \*\*FCL\*\* — библиотека, которая содержит огромный набор готовых к использованию классов. Например, если вам нужно считать данные из файла, вы можете воспользоваться классом `File`, который предоставляет нужные методы. \*\*BCL (Base Class Library)\*\* — это базовый набор классов FCL, такие как `System.String`, `System.Int32`.

## - \*\*CLI (Common Language Infrastructure)\*\* — стандарт, определяющий, как языки программирования могут взаимодействовать в рамках .NET. Этот стандарт позволяет создавать приложения на разных языках, которые будут совместимы между собой.

## - \*\*IL (Intermediate Language)\*\* — это язык промежуточного уровня, в который компилируются программы. Затем CLR преобразует этот IL в машинный код с помощью JIT-компилятора.

## 3. \*\*Пояснить работу JIT-компилятора?\*\*

## JIT (Just-In-Time) компилятор работает во время выполнения программы. Когда вы запускаете программу на .NET, сначала она компилируется в IL-код. Однако IL-код не может выполняться напрямую процессором. JIT-компилятор берет IL-код и переводит его в машинный код, который уже может выполняться на конкретном процессоре. Этот процесс происходит "по ходу" выполнения программы, что позволяет ускорить запуск и адаптировать код к разным системам.

## 4. \*\*Что такое CTS (Common Type System)?\*\*

## CTS определяет, как различные языки .NET должны описывать и использовать типы данных. Это важно, потому что .NET поддерживает несколько языков программирования (например, C#, VB.NET, F#), и чтобы код на одном языке мог взаимодействовать с кодом на другом, они должны иметь общие стандарты для описания типов данных. Например, целое число в C# (`int`) и в VB.NET (`Integer`) на уровне CLR представляют один и тот же тип данных — `System.Int32`.

## 5. \*\*Какие аспекты поведения определяет тип System.Object?\*\*

## `System.Object` — это базовый класс для всех типов в .NET. Любой тип, будь то примитивный тип вроде `int` или пользовательский класс, наследуется от `Object`. Он определяет базовые методы, такие как:

## - `ToString()` — возвращает строковое представление объекта.

## - `Equals()` — проверяет равенство объектов.

## - `GetHashCode()` — возвращает хэш-код объекта.

## - `GetType()` — возвращает тип объекта.

## 6. \*\*Что находится в mscorlib.dll?\*\*

## `mscorlib.dll` — это одна из самых важных библиотек в .NET, которая содержит основные классы, такие как `System.Object`, `System.String`, `System.Int32` и т.д. Это ядро .NET Framework, от которого зависят все другие библиотеки и компоненты.

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

## Сборка — это единица развертывания и выполнения приложения. Это скомпилированный код (например, в формате .exe или .dll), который включает в себя:

## - \*\*IL-код\*\* (код, который будет выполняться).

## - \*\*Метаданные\*\*, которые содержат информацию о классах, методах и других элементах программы.

## - \*\*Манифест\*\*, который описывает сборку (ее версию, зависимости и т.д.).

## 8. \*\*Какие виды сборок существуют?\*\*

## - \*\*Частные сборки\*\* — используются только внутри одного приложения.

## - \*\*Общие сборки\*\* — хранятся в GAC (Global Assembly Cache) и могут использоваться несколькими приложениями.

## - \*\*Сборки спутников\*\* — содержат локализованные ресурсы (например, строки для разных языков).

## 9. \*\*Что такое assembly manifest?\*\*

## Манифест сборки — это часть метаданных, которая содержит информацию о сборке: ее название, версию, зависимости от других сборок, а также информацию о безопасности. Манифест — это "паспорт" сборки, который помогает системе правильно ее использовать.

## 10. \*\*Что такое GAC?\*\*

## GAC (Global Assembly Cache) — это глобальное хранилище для общих сборок. Если сборка может использоваться разными приложениями, ее можно поместить в GAC, и тогда все приложения смогут к ней обращаться. Это экономит место и упрощает обновление версий сборок.

## 11. \*\*Чем managed code отличается от unmanaged code?\*\*

## - \*\*Managed code\*\* — это код, который работает под управлением CLR. CLR управляет памятью (с помощью сборщика мусора), обеспечивает безопасность и другие функции. Программы, написанные на C#, VB.NET и других языках .NET, являются managed code.

## - \*\*Unmanaged code\*\* — это код, который выполняется вне контроля CLR, как, например, программы на C++ без .NET. Такой код работает напрямую с памятью и требует ручного управления ресурсами.

## 12. \*\*Как и для чего определен метод Main?\*\*

## `Main` — это точка входа в любую программу на C#. Когда вы запускаете программу, именно с метода `Main` начинается выполнение кода. Внутри `Main` можно вызвать другие методы, создать объекты и управлять потоком программы. Этот метод может возвращать `void` или `int`, и в некоторых случаях принимать аргументы (например, параметры командной строки).

13. \*\*Варианты использования директивы `using` (using Directive) в C#?\*\*

Директива `using` в C# имеет два основных применения:

- \*\*Подключение пространств имен:\*\*

Например, `using System;` подключает пространство имен `System`, которое содержит классы, такие как `Console`. Это позволяет вам писать код, не указывая полные имена классов.

- \*\*Управление ресурсами:\*\*

Конструкция `using` используется для автоматического освобождения ресурсов. Например, при работе с файлами или подключениями:

```csharp

using (StreamReader reader = new StreamReader("file.txt"))

{

string line = reader.ReadLine();

}

```

После выполнения блока `using` ресурс автоматически закрывается и освобождается.

14. \*\*Как связаны между собой сборки и пространства имен?\*\*

Пространства имен организуют классы и типы в логические группы, облегчая их использование и предотвращая конфликты имен. В рамках одной сборки может быть множество пространств имен. Например, сборка может содержать как пространство имен `System.Collections`, так и `System.IO`. Сборка — это физическая единица кода (файл `.dll` или `.exe`), а пространства имен — это логическая организация внутри кода.

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

Примитивные типы данных — это базовые типы, встроенные в язык. Они хранят простые значения и не содержат сложных структур. Примеры примитивных типов:

- `int` — целое число.

- `double` — число с плавающей точкой.

- `char` — символ.

- `bool` — логический тип (true/false).

- `byte`, `sbyte`, `short`, `ushort`, `long`, `ulong`, `float`.

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

Ссылочные типы хранятся в куче, а переменная хранит ссылку на объект в памяти. Основные ссылочные типы:

- \*\*Классы\*\* (`class`) — любой пользовательский класс.

- \*\*Строки\*\* (`string`).

- \*\*Массивы\*\*.

- \*\*Интерфейсы\*\*.

Когда вы присваиваете ссылочный тип другой переменной, копируется ссылка на объект, а не сам объект.

17. \*\*Какие типы относятся к типам-значениям?\*\*

Типы-значения хранят данные непосредственно в переменной. Они выделяются в стеке и передаются по значению. Примеры:

- \*\*Примитивные типы\*\* (`int`, `double`, `bool`).

- \*\*Структуры\*\* (`struct`).

- \*\*Перечисления\*\* (`enum`).

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

- \*\*Значимые типы\*\* хранятся в стеке, и когда они передаются в методы, копируется само значение. Это значит, что изменения внутри метода не затронут оригинальную переменную.

- \*\*Ссылочные типы\*\* хранятся в куче, а переменная содержит ссылку на объект. Если передать ссылочный тип в метод, то изменения в методе повлияют на оригинальный объект, так как метод работает с той же ссылкой.

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

- \*\*Упаковка (boxing)\*\* — процесс преобразования значимого типа (например, `int`) в объект, чтобы он мог быть обработан как ссылочный тип. Пример:

```csharp

int x = 10;

object obj = x; // упаковка

```

- \*\*Распаковка (unboxing)\*\* — это обратный процесс, когда объект преобразуется обратно в значимый тип:

```csharp

int y = (int)obj; // распаковка

```

20. \*\*В чем заключается разница между `int` и `System.Int32`?\*\*

Разницы нет. `int` — это сокращение (alias) для `System.Int32`. Они обозначают одно и то же: целое число с 32-битным представлением.

21. \*\*Для чего используется тип `dynamic`?\*\*

`dynamic` позволяет отложить проверку типов до времени выполнения. Это удобно, когда тип данных может быть неопределен на этапе компиляции или когда нужно работать с COM-объектами или динамическими типами данных, такими как JSON или XML:

```csharp

dynamic obj = GetDynamicObject();

obj.DoSomething(); // проверка только во время выполнения

```

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

- \*\*`var`\*\* — тип выводится на этапе компиляции, и после этого тип переменной остается неизменным. Например, если присвоить `var x = 10;`, то `x` всегда будет `int`.

- \*\*`dynamic`\*\* — тип переменной определяется во время выполнения, и компилятор не проверяет корректность вызовов методов или доступа к свойствам. Ошибки могут возникнуть только во время выполнения программы.

23. \*\*Что такое неявно типизированная переменная?\*\*

Это переменная, тип которой определяется компилятором на основе присвоенного значения. Используется с ключевым словом `var`. Например:

```csharp

var number = 42; // компилятор выводит, что number — это int

```

24. \*\*Для чего используют Nullable тип?\*\*

Nullable типы (`int?`, `double?`) позволяют значимым типам (например, `int`, `bool`) принимать значение `null`. Это полезно, когда нужно указать, что значение переменной может отсутствовать:

```csharp

int? age = null; // переменная может содержать либо значение, либо null

```

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

Строки объявляются с использованием двойных кавычек: `"Hello, world!"`. Строки — неизменяемый тип, и с ними можно выполнять такие операции, как:

- Конкатенация (объединение строк): `string result = "Hello" + " World";`.

- Получение подстроки: `string sub = "Hello".Substring(0, 2); // "He"`.

- Сравнение строк: `if (str1 == str2)`.

26. \*\*Какие есть способы для задания и инициализации строк?\*\*

- Через литерал: `string str = "Hello";`.

- Через конструктор: `string str = new string('a', 5); // "aaaaa"`.

- С помощью интерполяции строк (начиная с C# 6.0): `string name = $"Hello, {userName}!";`.

27. \*\*Какие методы есть у типа String?\*\*

- `Length` — возвращает длину строки.

- `Substring()` — возвращает подстроку.

- `IndexOf()` — возвращает индекс первого вхождения символа.

- `ToUpper()` / `ToLower()` — преобразуют строку в верхний/нижний регистр.

- `Replace()` — заменяет одну подстроку на другую.

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

- Пустая строка (`""`) — это строка длиной 0, которая фактически существует в памяти.

- `null` — это отсутствие объекта, это значит, что переменная не указывает ни на какой объект в памяти.

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

- С помощью оператора `==`:

```csharp

if (str1 == str2) { }

```

- Метод `Equals()`:

```csharp

str1.Equals(str2);

```

- Метод `String.Compare()`:

```csharp

int result = String.Compare(str1, str2);

```

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

`String` — это неизменяемый тип. Каждый раз, когда строка изменяется, создается новый объект. Это неэффективно при множественных изменениях строки.

`StringBuilder` — это изменяемый тип, который позволяет изменять строку без создания новых объектов, что делает его более эффективным для частых операций над строками:

```csharp

StringBuilder sb = new StringBuilder("Hello");

sb.Append(" World");

```

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

Явные преобразования позволяют преобразовать один тип в другой. Например, чтобы преобразовать строку в число:

```csharp

string str = "123";

int number = Convert.ToInt32(str);

```

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

- Ввод:

```csharp

string input = Console.ReadLine();

```

- Вывод

:

```csharp

Console.WriteLine("Hello, world!");

```

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

- Одномерный массив:

```csharp

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

```

- Двумерный массив:

```csharp

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

```

34. \*\*Что такое ступенчатый массив? Как его задать?\*\*

Ступенчатый массив — это массив массивов, где каждая строка может иметь разную длину:

```csharp

int[ ][ ] jaggedArray = new int[3][ ];

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

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

jaggedArray[2] = new int[] { 6 };

```

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

В `foreach` можно использовать любые типы, которые реализуют интерфейс `IEnumerable`. Пример с массивом:

```csharp

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

foreach (int number in numbers)

{

Console.WriteLine(number);

}

```

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

Кортеж — это структура данных, которая позволяет объединить несколько значений разных типов в одну сущность. Пример:

```csharp

(int, string) person = (1, "John");

Console.WriteLine(person.Item1); // вывод: 1

```

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

Локальная функция — это функция, определенная внутри другой функции. Она доступна только внутри родительской функции:

```csharp

void OuterFunction()

{

void LocalFunction()

{

Console.WriteLine("Это локальная функция");

}

LocalFunction();

}

```

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

В блоке `checked` происходит проверка на переполнение арифметических операций (например, при переполнении диапазона типа), и если переполнение случится, то выбрасывается исключение. В блоке `unchecked` такая проверка отключается:

```csharp

checked

{

int x = int.MaxValue;

x++; // выбросит исключение

}

```

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

По умолчанию для большинства арифметических операций используется `unchecked` контекст, то есть переполнение игнорируется. Можно явно задать проверку с помощью блока `checked` или ключевых слов в коде:

```csharp

int x = checked(a + b); // явная проверка

```

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

Ключевое слово `fixed` используется для закрепления переменной в памяти (то есть чтобы указатель на объект не изменился во время работы сборщика мусора). Это нужно при работе с небезопасным кодом и указателями:

```csharp

fixed (int\* p = &x)

{

// код с указателем

}

```

**Приведенные здесь и далее теоретические сведения не являются достаточными для освоения тем (это краткий вводный материал). Необходимо использовать дополнительную литературу!!!!!**

Язык программирования C# является прямым наследником языка С++. Он унаследовал многие синтаксические конструкции языка С и объектно-ориентированную модель С++. В отличие от С++ С# является чисто объектно-ориентированным языком. В объектно-ориентированном программировании ход выполнения программы определяется объектами. Объекты это экземпляры класса. Класс это абстрактный тип данных, определяемый пользователем ( программистом). Класс включает в себя данные и функции для обработки этих данных. В С# запрещены глобальные функции. Все функции должны быть обязательно определены внутри класса. Не является исключением и главная функция языка С# Main( ) (в отличии от языка С пишется с прописной буквы).

Объявление класса синтаксически имеет следующий вид:

сlass имя\_класса

{

// члены класса

}

Члены класса это данные и функции для работы с этими данными. Рассмотрим шаблон приложения, подготовленный для нас мастером:

using System;

namespace ConsoleApplication10

{

/// <summary>

/// Summary description for Class1.

/// </summary>

class Class1

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main(string[] args)

{

//

// TODO: Add code to start application here

//

}

}

}

Первая строчка проекта *using System;*, включает в себя директиву *using*, которая сообщает компилятору, где он должен искать классы (типы), не определенные в данном пространстве имен. Мастер, по умолчанию, указывает стандартное пространство имен *System*, где определена большая часть типов среды .NET.

Следующей строчкой *namespace ConsoleApplication10* мастер предложения определяет пространство имен для нашего приложения. По умолчанию в качестве имени выбирается имя проекта. Область действия пространства имен определяется блоком кода, заключенного между открывающей и закрывающей фигурными скобками. Пространство имен обеспечивает способ хранения одного набора имен отдельно от другого. Имена, объявленные в одном пространстве имен не конфликтуют, при совпадении, с именами, объявленными в другом пространстве имен.

В шаблоне приложения имеется множество строк, которые являются комментариями.

В C# определены три вида комментариев:

* многострочный (/\*…\*/)
* однострочный (//…)
* XML (///) – комментарий для поддержки возможности создания самодокументированного кода.

Строчка [STAThread] является атрибутом. Атрибуты задаются в квадратных скобках. С помощью атрибута в программу добавляется дополнительная описательная информация, связанная с элементом кода, непосредственно перед которым задается атрибут. В нашем случае указывается однопоточная модель выполнения функции Main. Заголовок функции:

static void Main(string[] args)

Функция Main определена как статическая (static) с типом возвращаемого значения void. Функция Main( ) C# как и функция main( ) языка С может принимать аргументы. Аргумент - это строковый массив, содержащий элементы командной строки. Тело функции пустое и в нем содержится, в виде комментария, предложение добавить туда код для запуска приложения:

// TODO: Add code to start application here

Воспользуемся этим предложением и добавим в тело функции одну строчку:

static void Main(string[] args)

{

//

// TODO: Add code to start application here

**Console.WriteLine("Привет!");**

//

}

Функции консольного ввода-вывода являются методами класса Console библиотеки классов среды .NET.

Для ввода строки с клавиатуры используется метод Console.ReadLine(), а для ввода одного символа метод Console.Read().

Для консольного вывода также имеются две метода

* метод Console.Write(), который выводит параметр, указанный в качестве аргумента этой функции, и
* метод Console.WriteLine(),который работает так же, как и Console.Write(), но добавляет символ новой строки в конец выходного текста.

Для анализа работы этих методов модифицируйте функцию Main( ) так, как показано ниже :

static void Main(string[] args)

{

//

// TODO: Add code to start application here

**Console.WriteLine("Введите ваше имя");**

**string str=Console.ReadLine();**

**Console.WriteLine("Привет "+str+"!!!");**

**Console.WriteLine("Введите один символ с клавитуры");**

**int kod=Console.Read();**

**char sim=(char)kod;**

**Console.WriteLine("Код символа "+sim+" = "+kod);**

//

}

Добавим

**Console.WriteLine("Код символа {0} = {1}",sim,kod);**

Первым параметром списка является строка, содержащая маркеры в фигурных скобках. Маркер это номер параметра в списке. При выводе текста вместо маркеров будут подставлены соответствующие параметры из остального списка. После маркера через запятую можно указать, сколько позиций отводится для вывода значений. Например, запись {1,3} означает, что для печати первого элемента списка отводится поле шириной в три символа. Причем, если значение ширины положительно, то производится выравнивание по правому краю поля, если отрицательно то по левому.

Добавим 4 новые строчки в конец кода функции Main():

**int s1=255;**

**int s2=32;**

**Console.WriteLine(" \n{0,5}\n+{1,4}\n-----\n{2,5}",s1,s2,s1+s2);**

**Console.WriteLine(" \n{1,5}\n+{0,4}\n-----\n{2,5}",s1,s2,s1+s2);**

//

Кроме того, после поля ширины через двоеточие можно указать форматную строку, состоящую из одного символа и необязательного значения точности.

Существует 8 различных форматов вывода:

* С – формат национальной валюты,
* D – десятичный формат,
* E – научный (экспоненциальный) формат,
* F – формат с фиксированной точкой,
* G – общий формат,
* N – числовой формат,
* P – процентный формат,
* X – шестнадцатеричный формат

Например, запись {2,9:C2} – означает, что для вывода второго элемента из списка, отводится поле шириной в 9 символов. Элемент выводится в формате денежной единицы с количеством знаков после запятой равной двум. При выводе результата происходит округление до заданной точности.