

Введение в C#

# Знакомство с языком C#

Используется интегрированная среда разработки Microsoft Visual Studio 2019

---



# На этом уроке

1. Узнаем, что такое интегрированная среда разработки и из чего она состоит.
2. Установим интегрированную среду разработки Visual Studio и познакомимся с её основными функциями.
3. Создадим проект в Visual Studio и рассмотрим его структуру.
4. Научимся работать с консольным вводом-выводом и напишем первую программу.

## Оглавление

[На этом уроке](#)

[Как будет проходить обучение](#)

[Уроки](#)

[Настройка среды](#)

[Среда разработки](#)

[История языка C# и платформы .NET Framework](#)

[Как создаются программы](#)

[Создание консольного проекта](#)

[Проверка кода](#)

[Автодополнение](#)

[Сборка и запуск проекта](#)

[Отладка приложений, отладчик, точка останова](#)

[Разбор кода консольных программ на примере HelloWorld](#)

[Консольный ввод-вывод](#)

[Шаблонные строки](#)

[Форматирование кода](#)

[Комментарии](#)

[Домашнее задание](#)

[Дополнительные материалы](#)

[Используемые источники](#)

# Как будет проходить обучение

## Уроки

Самым важным условием успешного прохождения курса будет присутствие на вебинарах в **реальном** времени. Поэтому постарайтесь скорректировать свой график. Возможно, некоторые занятия придётся пересмотреть в записи и не раз. Часто информация обрабатывается в подсознании и при втором или третьем просмотре вы начинаете замечать много новых нюансов. Обязательно выполняйте практические задания. [Здесь](#) собраны рекомендации по их выполнению.

## Настройка среды

Для написания кода на курсе и выполнения домашних заданий необходимо установить [Microsoft Visual Studio версии 2019 Community Edition](#).

## Среда разработки

Среди основных преимуществ использования Visual Studio в качестве среды разработки можно выделить подсветку синтаксиса, встроенные инструменты для сборки и отладки исходного кода, а также возможность генерации новых проектов на основе большого количества шаблонов. При работе она может занимать до 1 Гб оперативной памяти.

# История языка C# и платформы .NET Framework

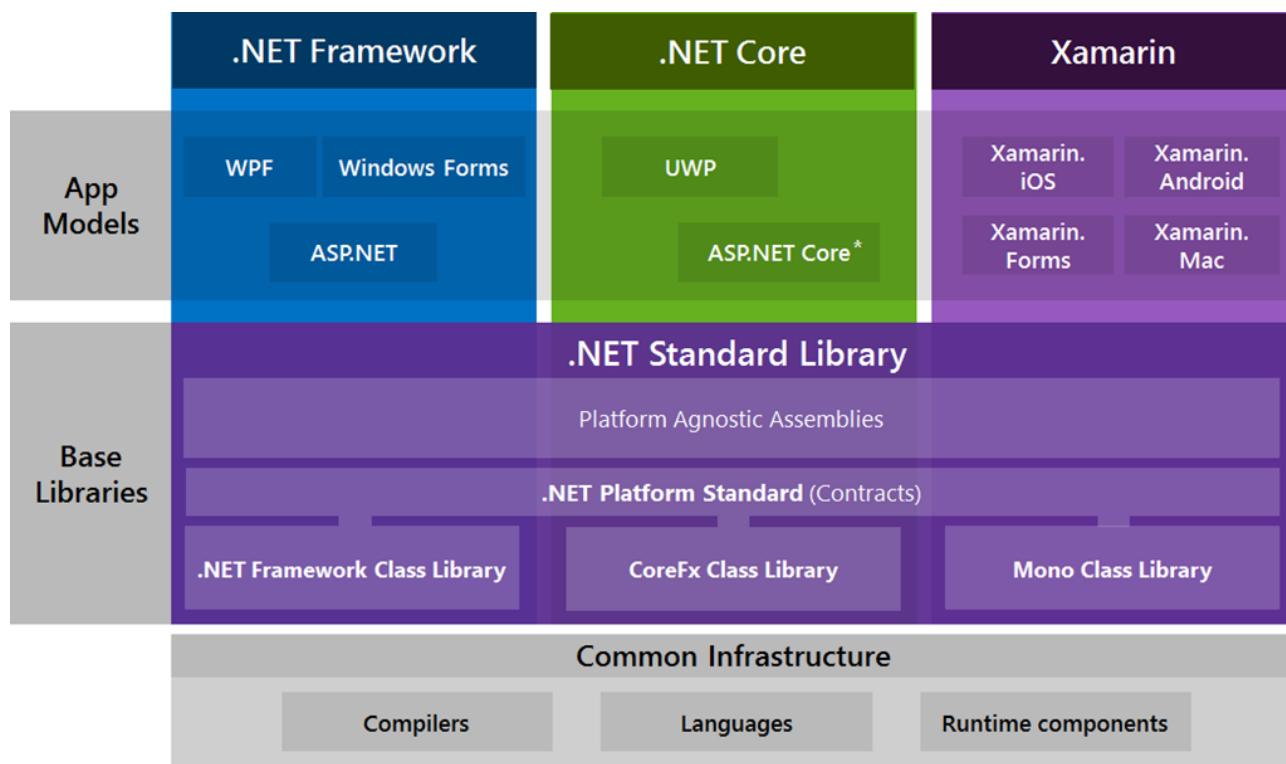
C# — это высокоуровневый язык разработки с 20-летней историей. Язык разработан компанией Microsoft и ранее применялся для написания прикладных консольных и графических приложений под ОС Windows. Сейчас на языке C# разрабатываются высоконагруженные облачные приложения, веб-сервисы, мобильные игры и всё те же консольные утилиты и прикладные графические приложения.

.NET Framework — это набор библиотек и утилит, позволяющих разрабатывать приложения, написанные на разных поддерживаемых языках (C#, F# и прочие) и запускать их в разных операционных системах. Несмотря на зрелость инфраструктуры, платформа .NET Framework и язык C# продолжают развиваться дальше и становятся более интересными и востребованными.

Чтобы комфортно погрузиться в образовательный процесс, а также упростить самостоятельный поиск информации, нам стоит разобраться с некоторыми основными терминами.

Так как C# и .NET применяются в различных сферах и могут быть использованы на разных операционных системах, со временем возникла необходимость создать определённый стандарт для

этих технологий, например, чтобы реализации .NET под ОС Windows, Linux и macOS обладали одинаковой функциональностью. Стандарт существовал не всегда и появился только порядка 5 лет назад. Но его появление очень сильно продвинуло технологии вперёд и дало возможность расширить реализации платформы .NET. Так как история языка C# и платформы .NET имеет разные этапы, сейчас доступны несколько реализаций платформы. Чтобы в дальнейшем вам было проще искать нужную информацию, стоит разобраться в наиболее часто встречающихся наименованиях из мира .NET.



.NET Standard — это спецификация платформы .NET. В ней описаны возможности различных стандартных библиотек. Спецификация служит своего рода документацией для тех команд, которые хотят разработать реализацию .NET. Например, если в стандарте описаны какие-либо операции с файлами, консолью или сетью, то они обязательно должны быть включены в реализацию. На основе .NET Standard нельзя создать консольные приложения<sup>1</sup>, но мы можем создать библиотеки классов, которые затем будем использовать в подходящих версиях .NET Core / .NET Framework

.NET Framework — самая первая реализация платформы .NET для ОС Windows. Актуальна до сих пор. Так как это первая реализация платформы, она существует со времён, когда .NET Standard еще не существовал. В итоге многие вещи из этой реализации перешли в стандарт. Но кроме стандартных возможностей, .NET Framework содержит функциональность, доступную только для ОС Windows. Например, написание приложений с графическим интерфейсом. Мы начнём погружение в мир .NET со знакомства с .NET Framework, а на следующих курсах перейдём на .NET Core.

<sup>1</sup> Приложения, которые не имеют графического интерфейса и взаимодействуют с пользователем через ввод клавиатурных команд и вывод текстовых сообщений. Например, командная строка Windows

.NET Core — современная реализация стандарта .NET Standard. В отличие от .NET Framework, позволяет разрабатывать приложения для различных ОС (Windows, Linux, macOS), но ценой потери некоторых функций. Например, .NET Core сейчас не поддерживает разработку десктопных приложений.

Unity — игровой движок, позволяющий разрабатывать игры для более чем 25 платформ — Windows, Linux, macOS, Android, iOS, PlayStation, Xbox и многих других!

MonoDevelop и Xamarin — сторонние реализации .NET Framework для платформ, отличных от Windows. MonoDevelop позволяет разрабатывать приложения для ОС Linux, Windows, macOS, в то время как Xamarin нацелен на разработку приложений для мобильных устройств. Сейчас Xamarin находится под управлением компании Microsoft.

В зависимости от конечной цели стоит использовать ту или иную реализацию платформы .NET. Преимуществом является то, что все реализации предполагают использование языка C#, поэтому владение одним языком программирования даст возможность вести разработки в одном или нескольких направлениях — от консольных приложений и веб-сервисов до разработки игр и высоконагруженных систем. Конечно, каждое направление требует серьезной и основательной подготовки, но знания языка C# сделает дальнейшее обучение более простым.

Список популярных приложений и ресурсов, разработанных с применением C# и .NET:

1. [StackOverflow](#) — самый посещаемый сайт для программистов. StackOverflow — это сайт, на котором можно делиться знаниями из мира IT, находить решения возникающих во время разработки проблем. Здесь люди задают вопросы и получают ответы миллионы раз в сутки. StackOverflow — это один из множества сайтов открытой платформы [StackExchange](#) — сети тематических ресурсов, насчитывающих около 1.3 миллиарда просмотров ежемесячно. Нужно отметить, что StackExchange практически полностью написана на языке C#.
2. [Visual Studio Tips for Software Developers](#) — сайт с советами и подсказками по работе с Visual Studio.
3. [ShareX](#) — мощный инструмент для захвата экрана
4. Тем, кто интересуется разработкой игр, стоит взглянуть на список игр, разработанных на движке Unity, среди которых нашумевший мультиплеер Among Us, Life is Strange: Before the Storm, и много других бестселлеров.

## Как создаются программы

Программы, которые мы привыкли использовать каждый день, будь то веб-браузер или офисный пакет, написаны на том или ином языке программирования. Язык программирования — это набор правил и команд, с помощью которых программист описывает поведение будущей программы. Существует множество языков программирования. Некоторые из них популярны и имеют широкое

применение, а некоторые — узкоспециализированы. При желании можно написать собственный язык программирования и выполнять на компьютере разработанные с его использованием программы.

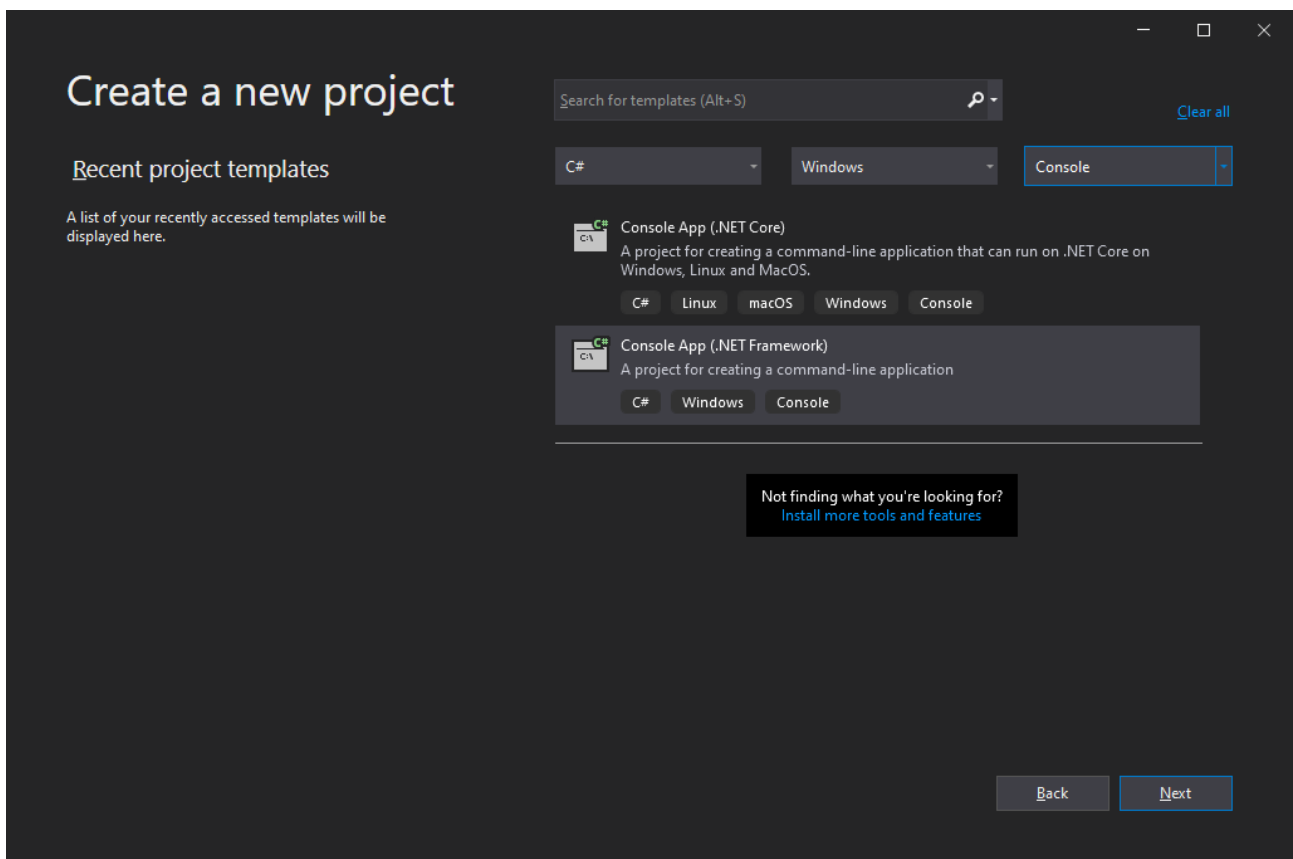
Но центральный процессор компьютера не знаком с понятием языков программирования. Мы можем запустить на одном и том же процессоре программы, написанные на разных языках. Это возможно благодаря тому, что процессор выполняет команды, состоящие из машинных кодов — набора числовых инструкций. Чтобы превратить написанную на языке программирования программу в набор машинных кодов, в зависимости от языка применяют различные специальные инструменты: линковщик, компилятор и прочие.

Чтобы упростить создание программ, разработчики используют интегрированные среды разработки — IDE (англ. Integrated Development Environment). IDE содержит инструменты для написания, сборки, тестирования и распространения программ. Зачастую функциональность IDE можно расширять с помощью сторонних дополнений. Для разработки на языке C# наиболее подходящая IDE — Microsoft Visual Studio. В курсе мы будем использовать версию, которую можно скачать бесплатно с официального сайта и использовать для написания полноценных приложений — Microsoft Visual Studio 2019 Community Edition.

## Создание консольного проекта

Запустите Visual Studio и выберите пункт Create a new project, найдите в списке Console App (.NET Framework) и нажмите Next. Для упрощения поиска можно задать значения для фильтров:

All languages => C#, All platforms => Windows, All project types => Console:



На следующем шаге укажите Project name — HelloWorld (Solution name обновится соответствующим образом) и нажмите Create. Созданный проект откроется в основном окне Visual Studio. Стоит отметить, что несмотря на то, что мы создавали проект, Visual Studio автоматически поместила его в решение.

**Решение (Solution)** — это набор проектов. Решения позволяют разбивать код больших приложений на более мелкие части, объединённые общей логикой. По умолчанию, когда мы создаём проект в Visual Studio, также создаётся решение с одноимённым названием. Решения имеют расширения файла `.sln`. Пока что мы сконцентрируемся на понимании проектов, и рассмотрим решения в следующих курсах.

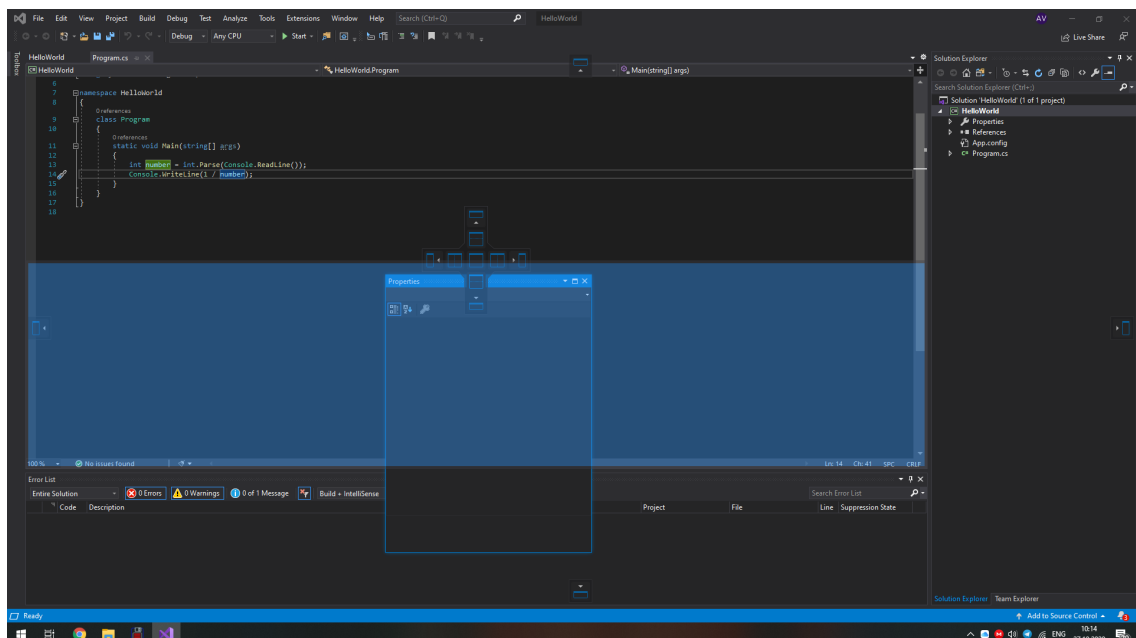
**Проект (Project)** — это набор исходных кодов, которые в результате сборки будут помещены в один и тот же выходной файл. Также проект содержит определённые сведения для компилятора, описание будущего исходного файла (например, его версию и издателя) и может содержать любой другой набор данных, необходимых приложению. Проект C# имеет расширение файла `.csproj`. По умолчанию файлы решения и проектов не отображаются в окне Visual Studio, т. к. зачастую нет необходимости редактировать их вручную.

Рассмотрим основные элементы окна Visual Studio и структуру проекта. Большую часть окна занимает область редактирования исходных кодов. Visual Studio, как и многие IDE, содержит встроенный текстовый редактор с подсветкой синтаксиса, индикацией ошибок, а также встроенную систему автодополнения текста IntelliSense (будет рассмотрена далее).

Справа от текстового редактора находятся панели Solution Explorer и Properties. Панель Properties отображает свойства выбранных элементов, и в ближайшее время мы не будем с ней работать. Рассмотрим панель Solution Explorer (Обозреватель решения).

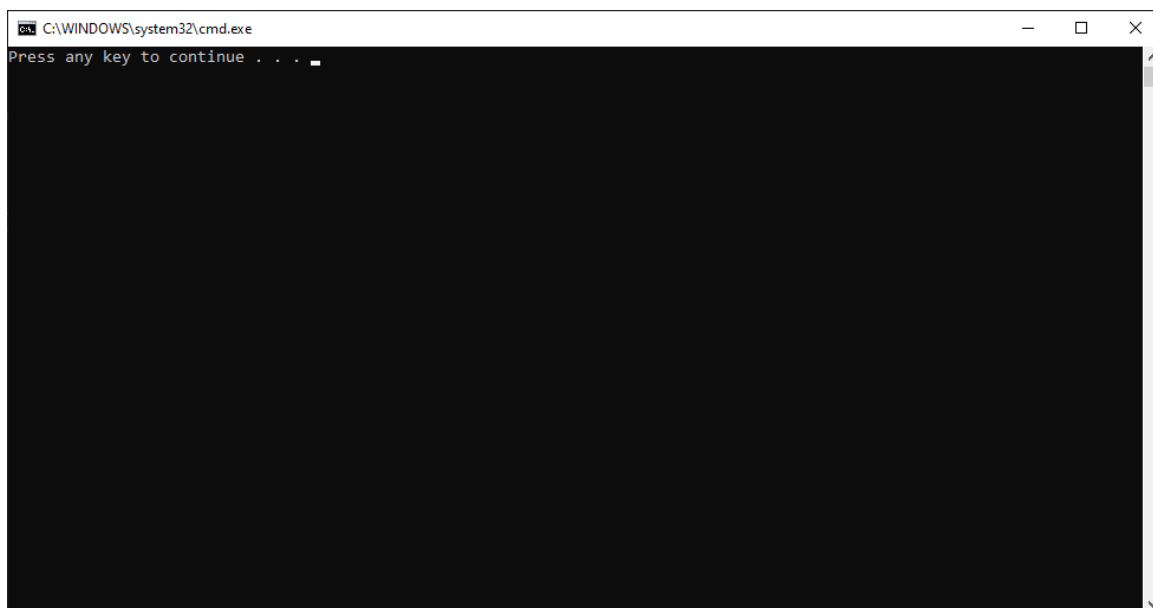
Solution Explorer отображает полную структуру нашего решения — список проектов, содержимое проектов, а также служебные элементы (Properties, References), которые позволяют управлять свойствами и зависимостями проекта (будут рассмотрены далее).

Вы можете настроить представление Visual Studio, например, изменив размер и расположение содержимого главного окна. Так же, как и обычные окна в Windows, панели Visual Studio можно перетащить за заголовок или прижать к краям главного окна. Если начать перетаскивание, на экране появится подсказка с активными значками базовых расположений. Если, не отпуская перетаскиваемый объект, навести мышью на какой-либо из значков, на экране появится заливка, означающая результирующее расположение объекта. Чтобы разместить объект этим образом, отпустите кнопку мыши на выбранном значке:



Рассмотрим подробнее файл Program.cs. Файлы с расширением .cs хранят код, написанный на языке C#. Файл Program содержит функцию Main. Мы остановимся на функциях в следующих уроках, а сейчас отметим, что выполнение программы в языке C# начинается с команд, написанных в функции Main. Сейчас функция Main не содержит команд, поэтому, если мы запустим программу, ничего не произойдёт. В этом легко убедиться. Запустим программу, нажав сочетание клавиш Ctrl + F5 — это сочетание производит сборку приложения и запуск без отладки. Мы увидим следующее окно:





Программа запустилась, но так как она пустая, ничего не произошло. Нажмите любую клавишу, чтобы закрыть консольное окно.

Чтобы программа начала выполнять какие-либо действия, нам нужно написать код в функции `Main`. Напишем нашу первую программу — `HelloWorld`. В программировании принято писать эту программу каждый раз, когда вы знакомитесь с новым языком или технологией. `HelloWorld` — простая программа, которая выводит в консоль строку с текстом `Hello, world!`. Несмотря на её простоту, написание такой программы даст нам понять, что происходит при сборке и запуске приложения.

Чтобы написать программу `HelloWorld`, добавьте строчку в тело функции `Main`, чтобы она выглядела следующим образом:

```
static void Main(string[] args)
{
    Console.WriteLine("Hello, world!");
}
```

Запустим программу знакомым сочетанием клавиш `Ctrl + F5`. В этот раз программа выведет на экран строку `Hello, world!` и после нажатия любой клавиши окно закроется. Несмотря на то, что мы смогли достаточно легко запустить программу, за кулисами IDE Visual Studio проделала некоторые сопутствующие этому операции:

1. Проверила, что код написан без ошибок.
2. Запустила компилятор языка C#, который собрал исходный код нашего проекта в двоичный (бинарный) исполняемый файл.
3. Запустила исполняемый файл нашего приложения, которое вывело на экран приветственное сообщение.

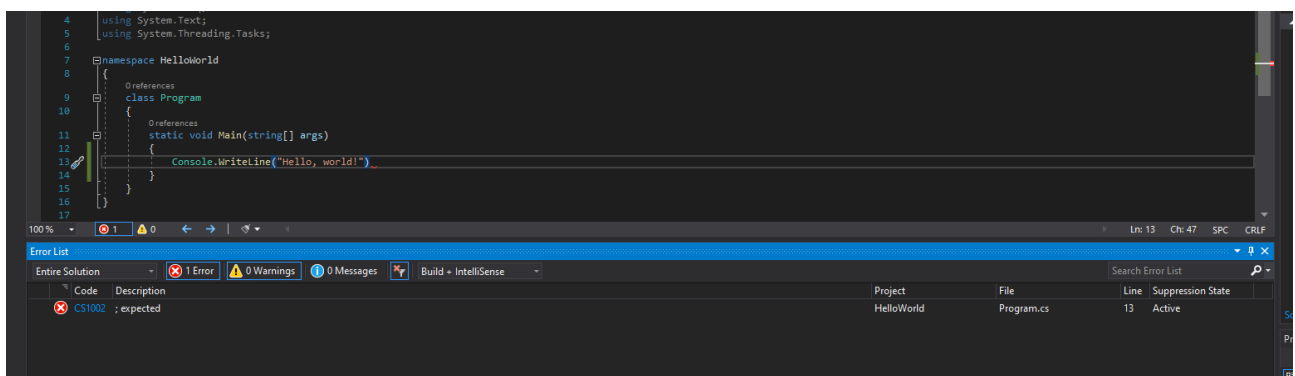
4. Дождалась нажатия клавиши от пользователя и закрыла окно с приложением.

Рассмотрим все эти шаги по очереди.

## Проверка кода

Пока мы пишем код, IDE производит с ним массу фоновых операций: проверяет на ошибки, ищет возможные способы его оптимизации, скачивает и устанавливает зависимости (при соответствующих настройках), а также составляет список из всевозможных функций, чтобы с помощью подсказок и автодополнения сделать процесс написания кода более удобным.

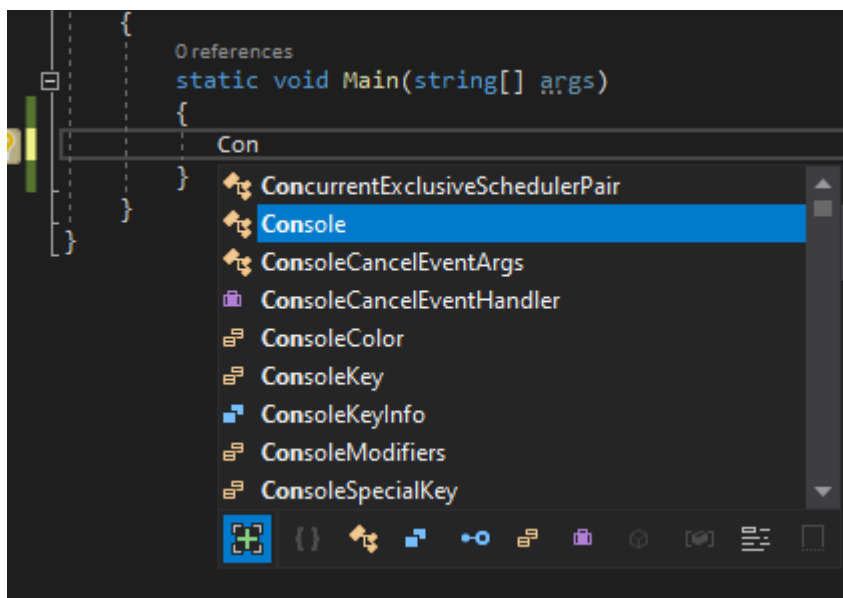
Сделаем ошибку в коде программы и убедимся, что она не проходит сборку. В языке C# принято разделять команды знаком точки с запятой. Уберем этот символ из команды, выводящей приветствие в консоль и попытаемся собрать проект сочетанием клавиш `Ctrl + Shift + B` — сборка проекта без запуска. Visual Studio подчеркнет красным место в коде, содержащее ошибку, а панель Error list внизу окна покажет ошибку. Если кликнуть на неё два раза, мы перейдём на строку, содержащую ошибку:



Как только мы добавим недостающий символ, ошибки исчезнут, так как теперь код стал безошибочным. Если мы снова допустим ошибку, Visual Studio сразу отобразит подсказку, так как она постоянно проводит проверку исходного кода.

## Автодополнение

Уберём из нашего кода строку, выводящую приветствие, и напишем ее еще раз. По мере набора кода Visual Studio будет отображать подсказки с именами команд, функций и переменных, подходящими под введенный нами текст. Это напоминает поисковую строку в браузере. Как только мы начинаем ввод первых символов, например `Con`, Visual Studio выберет слово `Console` как наиболее часто используемое. Чтобы не писать все слово вручную, мы можем нажать на `Tab` или `Enter` и тогда вводимая нами команда дополнится вариантом, который нам предложила IDE:

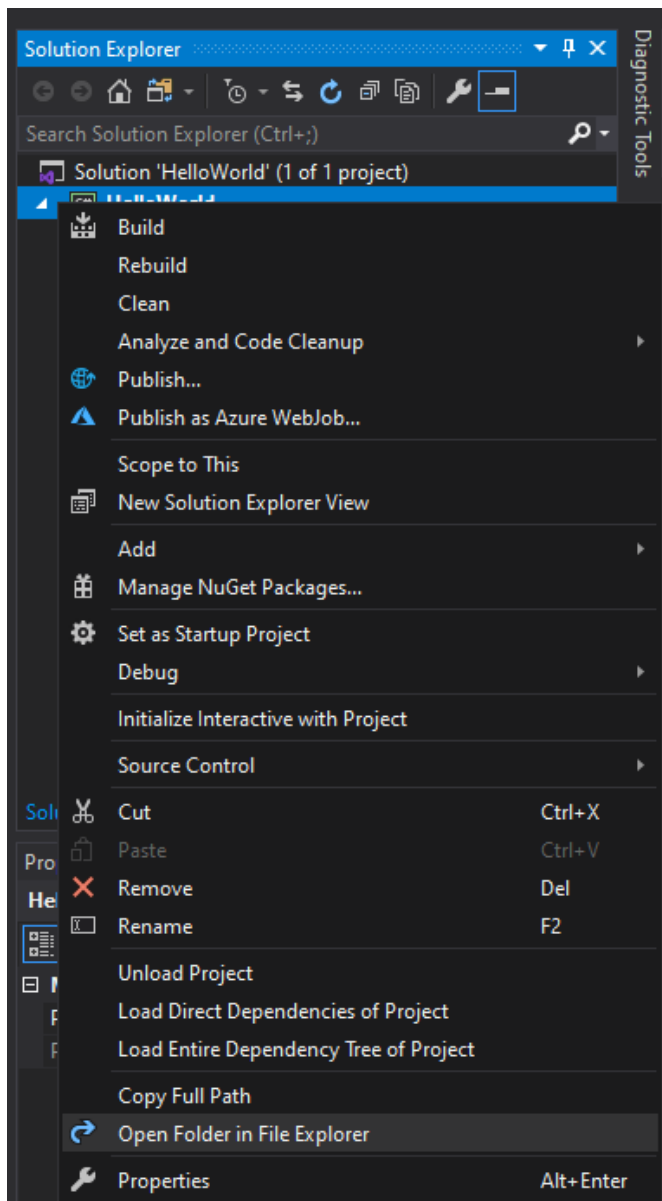


Если мы хотим выбрать другой вариант среди предложенных, мы можем перемещаться по ним клавиатурными стрелками и выбирать то, что нам нужно. После ввода части команды `Console.`, IDE вновь предложит нам наиболее часто применяемые функции. Подсказки автодополнения можно выбрать в любой момент сочетанием клавиш `Ctrl + Space`. Допишите команду и выведите на экран приветствие.

## Сборка и запуск проекта

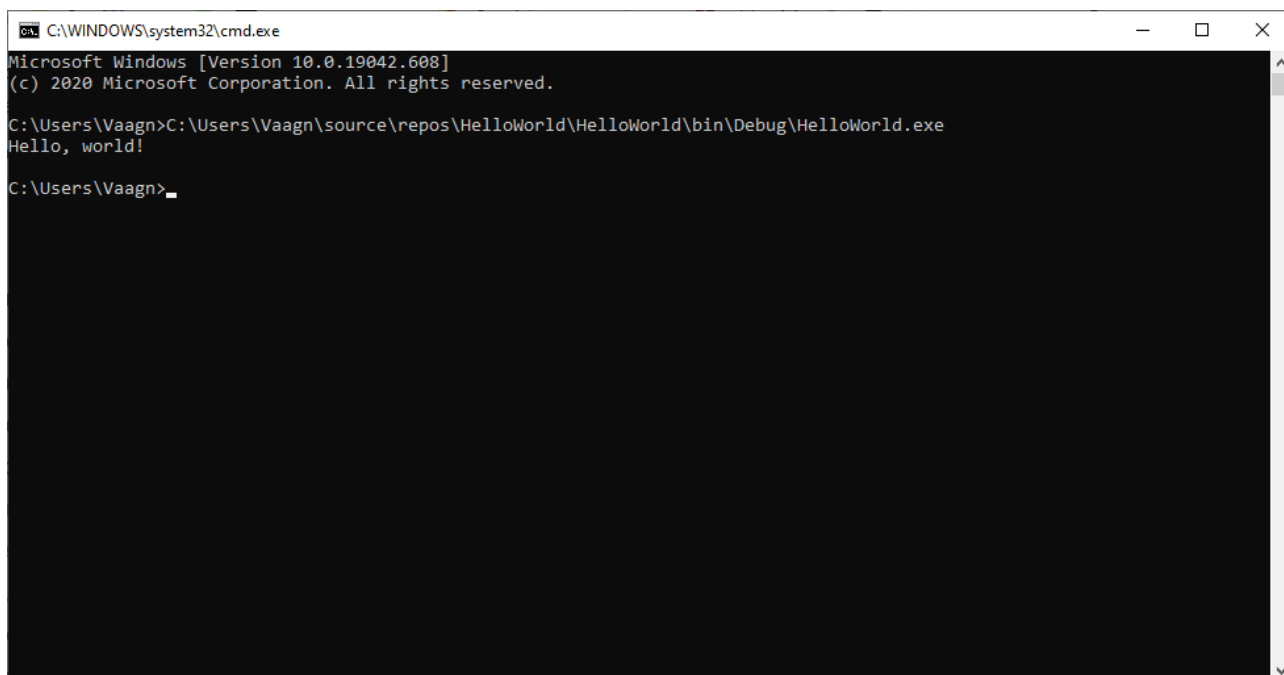
Рассмотрим, что происходит во время сборки проекта. Как упоминалось ранее, сборка — это процесс объединения исходных кодов и всех необходимых для работы данных приложения в выходной исполняемый файл. В ОС Windows исполняемые файлы чаще всего имеют расширение `.exe`. Мы уже видели, как Visual Studio запускает написанную нами программу, но что для этого необходимо предпринять?

Когда мы запускаем сборку проекта в Visual Studio (`Ctrl + Shift + B`) или само приложение (`Ctrl + F5`), Visual Studio создаёт исполняемый файл. Чтобы найти этот файл на диске, необходимо кликнуть правой кнопкой мыши по проекту в окне Solution Explorer и выбрать пункт `Open folder in Explorer`:



Далее в открывшемся каталоге необходимо перейти в папку `bin\Debug`.

Мы увидим исполняемый файл нашей программы и несколько служебных файлов. Если мы запустим программу, может показаться, что ничего не произошло. На самом деле программа запустилась, вывела приветствие и закрылась. При запуске программы из Visual Studio IDE запускает программу в окне терминала (обработчик консольных команд, встроенный в систему) и поэтому после завершения программы мы видим результат её выполнения. Чтобы увидеть, как работает исполняемый файл программы, мы можем также запустить его в терминале. Для этого нажмите сочетание клавиш `Windows + R` и в открывшемся диалоге `Run` (Выполнить) введите `cmd` и нажмите `Enter`. Перетащите в открывшееся окно терминала исполняемый файл программы и нажмите `Enter`:

A screenshot of a Windows Command Prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The window content shows the following text: "Microsoft Windows [Version 10.0.19042.608] (c) 2020 Microsoft Corporation. All rights reserved. C:\Users\Vaagn>C:\Users\Vaagn\source\repos\HelloWorld\HelloWorld\bin\Debug\HelloWorld.exe Hello, world! C:\Users\Vaagn>". The prompt is at the end of the last line.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.608]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Vaagn>C:\Users\Vaagn\source\repos\HelloWorld\HelloWorld\bin\Debug\HelloWorld.exe
Hello, world!

C:\Users\Vaagn>
```

Так мы можем убедиться, что исполняемый файл программы будет результатом сборки проекта и выполняет те команды, которые мы описали в исходном коде.

## Отладка приложений, отладчик, точка останова

В предыдущем разделе мы написали достаточно простую программу. Она не содержит логики или сложных вычислений и, глядя на её код, можно легко понять, что она делает. Зачастую при разработке программ допускаются логические ошибки. При этом код написан верно и программа успешно компилируется (собирается компилятором), но ведёт себя не так, как мы от неё ожидали.

Ошибки в программном обеспечении укрупнённо можно разбить на три основные категории:

1. **Ошибки времени проектирования** (их также называют ошибками времени компиляции, Design-time errors, Compile-time errors) — это опечатки, нарушение синтаксиса языка и прочие ошибки, при которых компилятор не может собрать программу. В примере с пропущенным знаком `;` в конце команды мы как раз совершили compile-time-ошибку.
2. **Ошибки времени выполнения (runtime errors)** — ошибки, которые сложно обнаружить без запуска программы. Возникают в случае, когда исходный код написан правильно и программа успешно собрана, но после запуска она сталкивается с серьёзной проблемой, обработка которой не была предусмотрена в коде. Например, мы написали программу, которая выводит на экран содержимое определённого файла, а файл оказался удалён. Другой пример: мы написали калькулятор и не предусмотрели деление на ноль; в таком случае программа может получить runtime-ошибку.
3. **Логические ошибки, бизнес-ошибки** — тот случай, когда программа успешно собрана и при её выполнении не возникает runtime-ошибок, но её поведение отличается от желаемого. Например, мы написали калькулятор, который не учитывает приоритет операций. В таком случае он будет выводить числовой ответ, который зачастую будет неправильным. С точки зрения компьютера,

программа работает корректно, так как она принимает на вход числа, производит над ними операции и выводит полученное число. Компьютеру сложно отследить логические ошибки, поэтому бизнес-ошибки наиболее коварные, они могут проявить себя не сразу, а спустя продолжительное время. И всё это время программа продолжит работать неправильно.

Важно понимать, что программа всегда делает не то, что мы от неё хотим, а то, что в ней написано. Один из способов разобраться с тем, что происходит во время выполнения программы — отладка. Отладка (от англ. debug) — процесс выявления ошибок (багов, bugs) в программном коде для их дальнейшего исправления. Об истории возникновения термина можно почитать [здесь](#).

Во время отладки программа запускается под управлением специальной программы — отладчика (debugger), которая даёт нам доступ к значениям переменных, объявленных в теле программы, позволяет приостанавливать её выполнение на определённых строках кода и при определённых условиях. Отладим приложение HelloWorld. Так как сейчас программа фактически состоит из одной строки, мы внесём в неё небольшие изменения.

Перепишем функцию Main так:

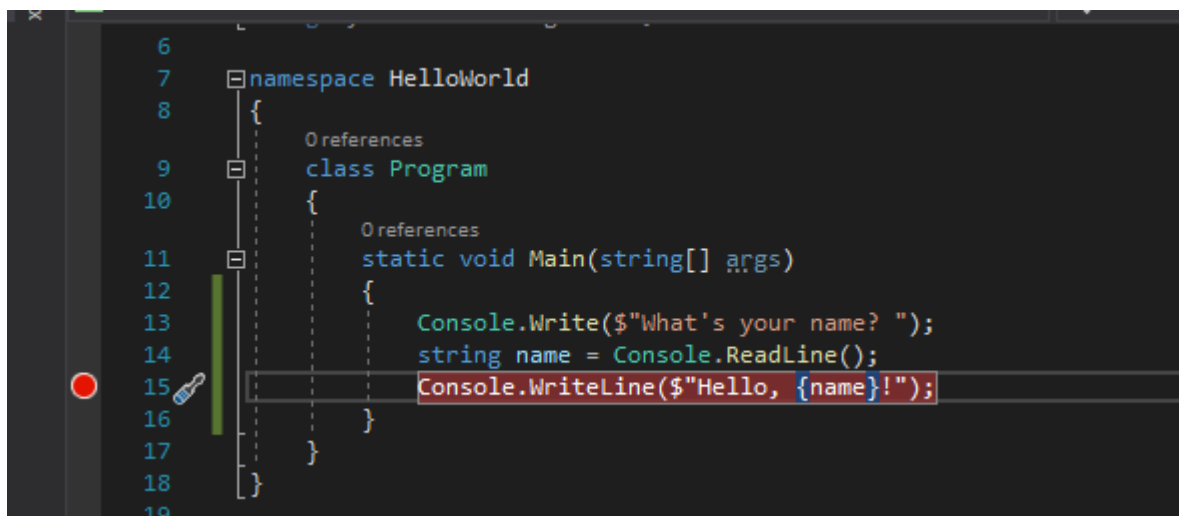
```
static void Main(string[] args)
{
    Console.Write("What's your name? ");
    string name = Console.ReadLine();
    Console.WriteLine($"Hello, {name}!");
}
```

Мы выводим сообщение в консоль и ожидаем ввода пользователя. Результат ввода мы получаем с помощью функции `Console.ReadLine`, которая возвращает всё, что ввёл пользователь до нажатия клавиши `Enter`. Введённая строка сохраняется в переменной `name`. Переменные хранят значения определённого типа. В данном случае тип переменной — `string`, это означает, что она может хранить произвольную строку. Затем мы выводим приветствие. Так как мы хотим вывести значение переменной, а не просто слово `name`, то добавили символ `$` в начало строки и обратились к переменной в фигурных скобках. Эта операция называется интерполяцией и о ней мы поговорим в разделе о строках.

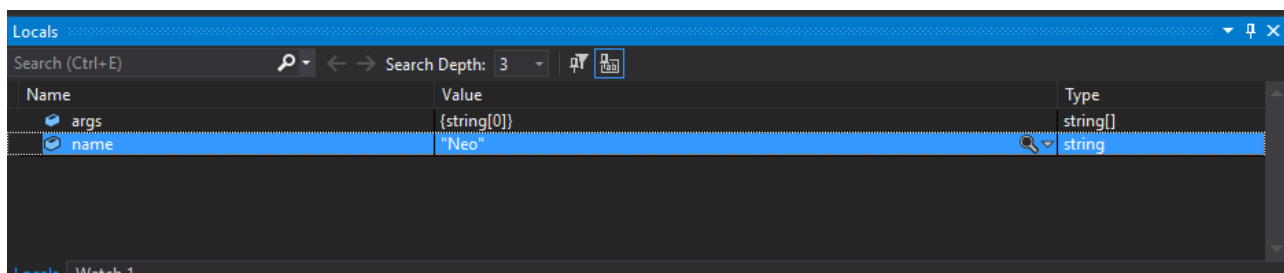
Запустим наш проект в режиме отладки — это означает, что Visual Studio соберёт новый исполняемый файл для нашей программы, а затем, вместо обычного запуска, передаст его во встроенный отладчик, благодаря чему мы сможем отлаживать HelloWorld внутри основного окна IDE. Чтобы запустить программу в режиме отладки, нажмите `F5`.

После запуска программа будет ожидать от нас ввода какой-либо строки в качестве имени. Введите строку и нажмите `Enter`. Как вы уже могли догадаться, наша программа выполнилась и успешно завершилась. По умолчанию, Visual Studio в режиме отладки приостанавливает выполнение программы только в случае возникновения ошибки. Чтобы приостановить выполнение программы на определённой строке кода (например, чтобы посмотреть значения имени, которое ввел пользователь) существуют точки останова (брейкпойнт, от англ. breakpoint). Перейдём на строку, выводящую

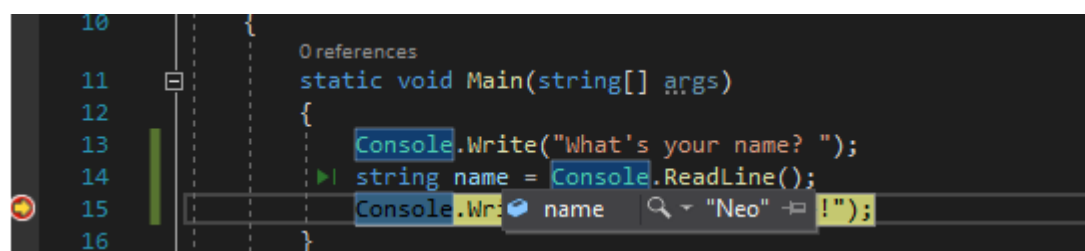
приветствие и нажмем клавишу установки/снятия точки останова — F9. Слева от текущей строки появится красный кружок, а сама строка получит красную заливку:



Вновь запустим программу в режиме отладки и укажем любое имя. После ввода имени выполнение программы приостановится на указанной строке, а интерфейс IDE немного изменится: в нижней части окна отобразится окно Locals, содержащее значения доступных переменных, и окно Call Stack, о котором мы поговорим позднее. В окне Locals мы увидим значение, указанное пользователем:



Также значения любых переменных, описанных до текущей строки, можно посмотреть наведением мыши:



Это значение появилось во время выполнения программы. Если программа ведет себя не так, как мы ожидали, мы можем проверить, действительно ли переменные содержат те значения, которые мы ожидаем. Это может существенно облегчить поиск и исправление ошибок в программном коде.

## Разбор кода консольных программ на примере HelloWorld

В этом разделе мы рассмотрим основные блоки консольной программы, написанной на языке C#. В качестве примера рассмотрим уже известную нам программу HelloWorld. Ранее мы не вдавались в подробности исходного кода этой программы и сфокусировались только на содержимом главной

функции `Main`. Теперь же мы подробно рассмотрим все составляющие этой программы. Все составляющие `HelloWorld`, за исключением кода функции `Main`, будут стандартными для консольных проектов `Visual Studio`, а это значит, что во всех последующие примеры будут иметь практически такое же содержимое. Поэтому стоит остановиться и разобраться в том, как всё работает.

Выполнение любой программы начинается с точки входа. Низкоуровнево, это адрес в памяти, с которого процессор начинает выполнять команды. В языках программирования существуют различные способы задать точку входа. В языке `C#` точкой входа в программу является функция следующего вида:

```
static void Main()
{
}
```

Тело функции и её аргументы не имеют значения. Главное — объявить функцию как `static void Main`. Определение для `static` и `void` будет дано в дальнейших разделах.

Программа на языке `C#` должна обязательно включать функцию `static void Main`, и при том, только одну. Иначе программу не получится собрать.

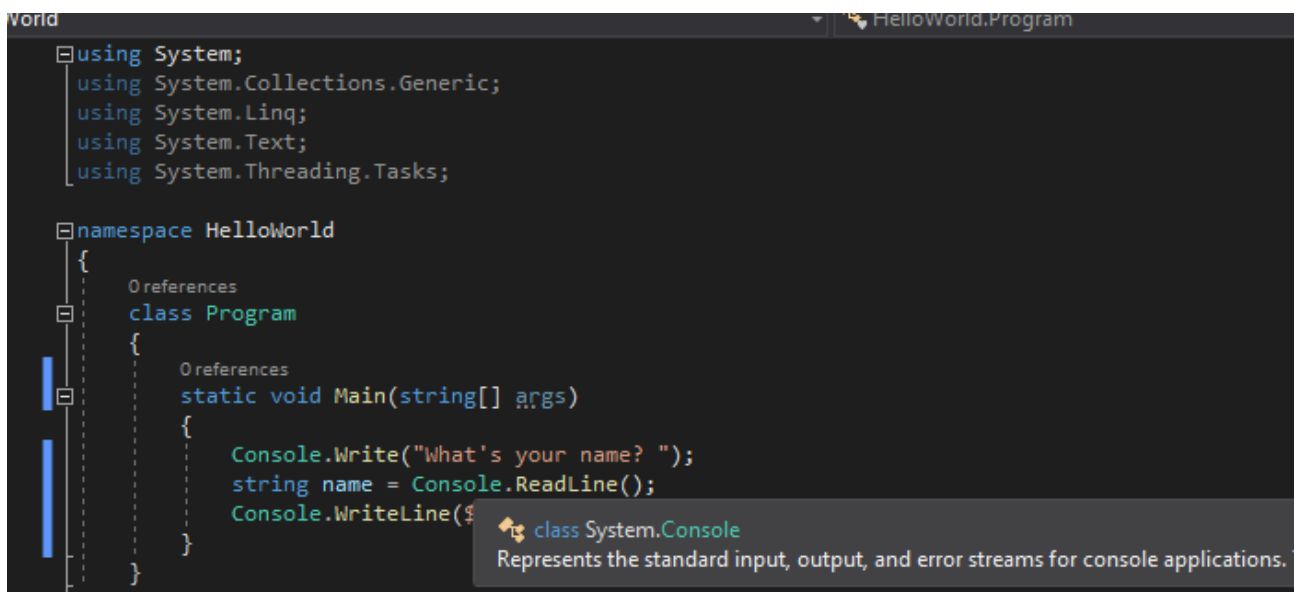
Мы разобрались с назначением функции `Main`. Но содержимое файла `Program.cs` начинается вовсе не с объявления точки входа. Рассмотрим исходный код этого файла:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("What's your name? ");
            string name = Console.ReadLine();
            Console.WriteLine($"Hello, {name}!");
        }
    }
}
```

В начале мы видим несколько строк, начинающихся с `using` — в данном контексте это оператор, позволяющий подключать пространства имён (`namespaces`, неймспейсы). Когда мы подключаем к своей программе пространство имён, мы получаем возможность использовать описанный в этом пространстве имён исходный код. Если мы внимательно посмотрим на содержимое `Program.cs`, мы заметим, что пространство имён `System` выделено белым шрифтом, в отличие от других. Всё потому что используемый нами класс `Console` находится в пространстве имён `System` — в этом легко убедиться, если навести на него мышкой:





Если мы удалим строку `using System;`, мы получим ошибку, так как теперь компилятору неизвестно ничего о классе `Console`.

Обратим внимание, что сам класс `Program` тоже описан внутри пространства имен — `HelloWorld`. Хотя это и не обязательно, но на практике все объявления принято делать внутри того или иного пространства имён. В противном случае у нас могут возникнуть сложности, если мы решим переиспользовать код в других местах.

Кроме логического разделения кода, пространства имён играют ещё одну важную роль — у типов, описанных внутри пространства имён, должны быть уникальные имена. Например, нельзя объявить два класса `Program` в пространстве имен `HelloWorld`. Типы и более продвинутое применение пространств имён мы рассмотрим в дальнейшем.

## Консольный ввод-вывод

Во времена, когда программы писались в машинных кодах, для вывода строки на экран необходимо было совершить ряд трудоемких операций с памятью — нужно было разместить строку в памяти, получить её адрес в памяти, затем положить это значение в определённые ячейки процессора и наконец, передать процессору команду на вывод строки на экран.

В предыдущих примерах мы считывали значения из консоли и выводили строки на экран, при этом нам не приходилось прилагать для этого какие-либо усилия, так как мы просто использовали возможности класса `Console`, который скрывает от нас сложности консольного ввода-вывода и предоставляет ряд функций для осуществления этих операций. Рассмотрим наиболее часто используемые из них:

`Console.ReadLine` возвращает строку, введённую пользователем, а затем переводит курсор на новую строку. Это означает, что последующие строки, выведенные на экран, будут размещены с новой строки. Пользовательский ввод завершается клавишей `Enter`.

`Console.WriteLine` выводит строку на экран и переводит курсор на новую строку.

`Console.ReadKey` возвращает информацию о нажатой пользователем клавише. Перевод на новую строку не осуществляется.

`Console.Read` возвращает информацию, о нажатой пользователем клавише и переводит консоль на новую строку. Пользовательский ввод завершается клавишей Enter.

`Console.Write` выводит строку на экран. Перевод на новую строку не осуществляется.

`Console.Clear` очищает содержимое консоли.

## Шаблонные строки

Ранее мы встречали команду, выводящую на экран приветствие пользователю:

```
Console.WriteLine($"Hello, {name}!");
```

В начале выводимой строки стоит специальный знак доллара (\$), который ставится перед шаблонными строками. Как следует из названия, шаблонные строки служат неким шаблоном, который можно переиспользовать с различным набором значений. Шаблонизация часто применяется в программировании. Например, проект консольного приложения HelloWorld создан IDE с помощью шаблонизации — в начале Visual Studio запрашивает у нас информацию о названии проекта и решения, а затем устанавливает эти значения в наборе шаблонных файлов. На выходе получается проект, содержащий заданные нами названия в качестве имени проекта, решения и пространства имён.

Шаблонизация строк имеет ещё одно название — **интерполяция**. А шаблонные строки чаще называются templates (шаблоны)

В теле шаблонных строк можно писать любые выражения, заключив их в фигурные скобки. Это означает, что мы можем подставить в такую строку значение переменной или результат выполнения какой-либо функции, или просто написать какое-нибудь арифметическое выражение. В любом случае, в строку подставится значение заданного выражения. Шаблонные строки зачастую применяют для форматированного вывода текста. Допустим, мы хотим вывести на экран следующую строку:

```
[28.10.2020 10:29:50] Hello, Neo!
```

То есть, вывести текущую дату и время в квадратных скобках, затем через пробел вывести слово Hello, указать имя пользователя и завершить строку восклицательным знаком. Чтобы вывести в консоль значения нескольких переменных, мы можем объединить их оператором сложения (+). Для строк такая операция называется конкатенацией, и мы рассмотрим её в следующих разделах. А пока что сравним вывод одной и той же строки сначала без применения шаблонных строк, а затем с их помощью:

```
Console.WriteLine "[" + DateTime.Now + "]" + " " + "Hello, " + name + "!";
```

```
Console.WriteLine($"{DateTime.Now}] Hello, {name}!");
```

Как видно из примера, использование интерполяции более читабельно, не требует ввода множества кавычек и операторов сложения, и нам не приходится расставлять пробелы внутри составляющих выводимой строки.

## Форматирование кода

В заключение этого урока, поговорим о внешнем виде нашего кода. Чтобы показать, какие участки кода находятся на одном логическом уровне, в языке C# применяются пробельные отступы. Вы могли обратить внимание на то, что у многих строчек кода HelloWorld имеются отступы слева. Причём эти отступы содержат разное количество символов. Отступы помогают разобраться в иерархии кода. Достаточно посмотреть на красиво оформленный код и мы сразу заметим, что класс `Program` находится в пространстве имен `HelloWorld`, и содержит в свою очередь функцию `Main`, чьё содержимое также сдвинуто левее объявления функции. Посмотрим на тот же исходный код, в котором не соблюдены отступы:

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("What's your name? ");
            var name = Console.ReadLine();
            Console.WriteLine($"{DateTime.Now}] Hello, {name}!");
        }
    }
}
```

Такой код полностью работоспособен, но его сложнее читать и в нём трудно разобраться. Для того, чтобы привести такой код в порядок, нужно его отформатировать — расставить отступы в нужном количестве. Visual Studio имеет функцию форматирования кода и нам достаточно нажать последовательно два сочетания клавиш: `Ctrl + K`, а затем `Ctrl + D`. Важно следить за форматированием кода, особенно в командной разработке.

## Комментарии

В коде программ на языке C# можно писать комментарии, пояснения или замечания. Комментарии бывают однострочные и многострочные.

Однострочные комментарии могут быть написаны как на новой строке, так и после определенной команды. В обоих случаях однострочным комментарием считается содержимое строки, начиная с двойного знака косой черты — /.

Примеры однострочных комментариев:

```
Console.WriteLine($"Hello, {name}!"); // выводим приветствие
```

```
// Выводим приветственное сообщение:  
Console.WriteLine($"Hello, {name}!");
```

Многострочные комментарии отделяются от кода последовательностью символом /\* в начале и \*/ в конце комментария. Также Visual Studio может вставлять символы \* в начале новой строки комментария:

```
/* Это пример многострочного комментария,  
 * написанного в Visual Studio  
 */  
Console.WriteLine($"Hello, {name}!");
```

Редактор кода выделяет комментарии зеленым цветом.

## Практическое задание

Написать программу, выводящую в консоль текст: «Привет, %имя пользователя%, сегодня %дата%». Имя пользователя сохранить из консоли в промежуточную переменную. Поставить точку останова и посмотреть значение этой переменной в режиме отладки. Запустить исполняемый файл через системный терминал.

## Дополнительные материалы

1. [Учебник по языку C# 9.0 и платформе .NET 5.](#)
2. [Руководство по повышению производительности Visual Studio.](#)
3. [Сочетания клавиш для клавиатуры и мыши. Visual Studio.](#)
4. [Использование Visual Studio без мыши.](#)

## Используемые источники

1. [Showcase of projects built with .net technology. BuiltWithDot.Net.](#)
2. [List of Unity games.](#)

3. [Performance.](#)
4. [Visual Studio Tips for Software Developers.](#)
5. [ShareX. Screen capture, file sharing and productivity tool.](#)
6. [Всё началось с мотылька | Блог ВКонтакте.](#)