# Введение в C#

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# уже не молодой язык и как и вся платформа .NET уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 11, которая вышла 8 ноября 2022 года вместе с релизом .NET 7.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей.

### Роль платформы .NET

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Как-то Билл Гейтс сказал, что платформа .NET - это лучшее, что создала компания Microsoft. Возможно, он был прав. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

* **Поддержка нескольких языков**. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому при определенных условиях мы можем сделать отдельные модули одного приложения на отдельных языках.
* **Кроссплатформенность**. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - .NET 7 поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.
* **Мощная библиотека классов**. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.
* **Разнообразие технологий**. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.
* К этому стоит добавить активной развивающийся и набирающий популяность Blazor - фреймворк, который работает поверх .NET и который позволяет создавать веб-приложения как на стороне сервера, так и на стороне клиента. А в будущем будет поддерживать создание мобильных приложений и, возможно, десктоп-приложений.
* **Производительность**. Согласно ряду тестов веб-приложения на .NET 7 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 7 в принципе отличаются высокой производительностью.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

### .NET Framework и .NET 7

Стоит отметить, что .NET долгое время развивался премущественно как платформа для Windows под названием .NET Framework. В 2019 вышла последняя версия этой платформы - .NET Framework 4.8. Она больше не развивается

С 2014 Microsoft стал развивать альтернативную платформу - .NET Core, которая уже предназначалась для разных платформ и должна была вобрать в себя все возможности устаревшего .NET Framework и добавить новую функциональность. Затем Microsoft последовательно выпустил ряд версий этой платформы: .NET Core 1, .NET Core 2, .NET Core 3, .NET 5. И текущей версией является расматриваемая в этом руководстве платформа .NET 7. Поэтому следует различать .NET Framework, который предназначен преимущественно для Windows, и кросплатформенный .NET 7. В данном руководстве речь будет идти о C# 11 в связке с .NET 7, поскольку это актуальная платформа.

### Управляемый и неуправляемый код

Нередко приложение, созданное на C#, называют **управляемым кодом** (managed code). Что это значит? А это значит, что данное приложение создано на основе платформы .NET и поэтому управляется общеязыковой средой CLR, которая загружает приложение и при необходимости очищает память. Но есть также приложения, например, созданные на языке С++, которые компилируются не в общий язык CIL, как C#, VB.NET или F#, а в обычный машинный код. В этом случае .NET не управляет приложением.

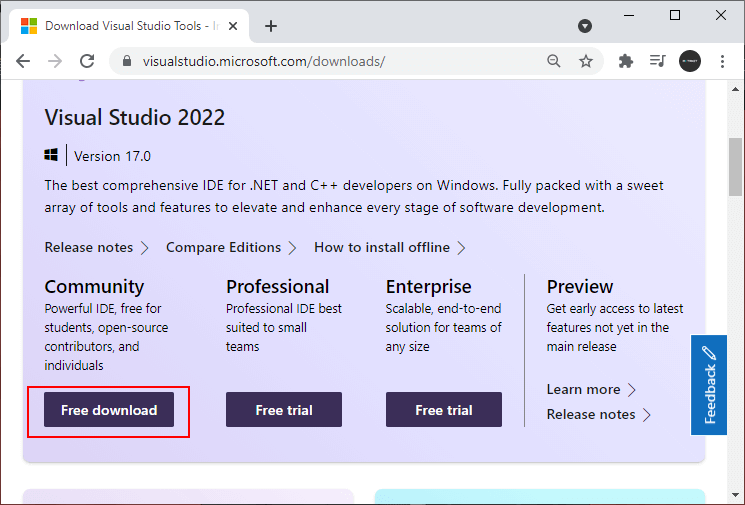
В то же время платформа .NET предоставляет возможности для взаимодействия с неуправляемым кодом..

### JIT-компиляция

Как выше писалось, код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку наше приложение может быть большим и содержать кучу инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

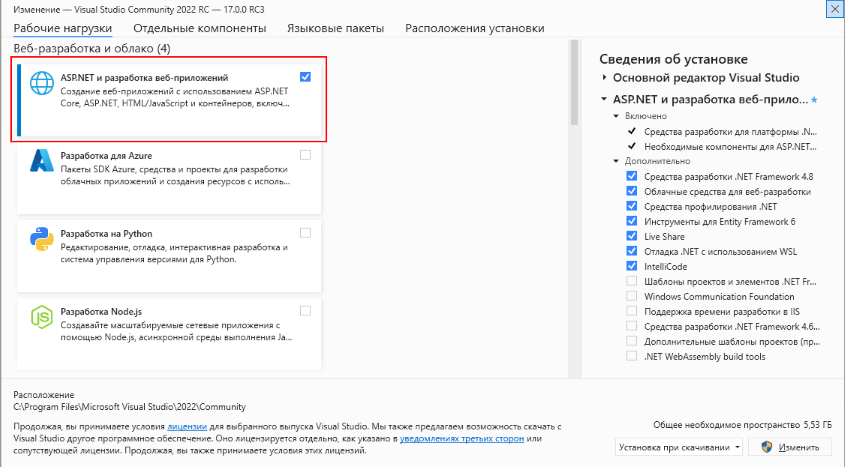
По сути это все, что вкратце надо знать о платформе .NET и языке C#. А теперь создадим первое приложение.

Чтобы облегчить написание, а также тестирование и отладку программного кода нередко используют специальные среды разработки, в частности, Visual Studio. Рассмотрим создание приложений на C# с помощью бесплатной и полнофункциональной среды Visual Studio Community 2022, которую можно загрузить по следующему адресу: [Microsoft Visual Studio 2022](https://visualstudio.microsoft.com/ru/downloads/)



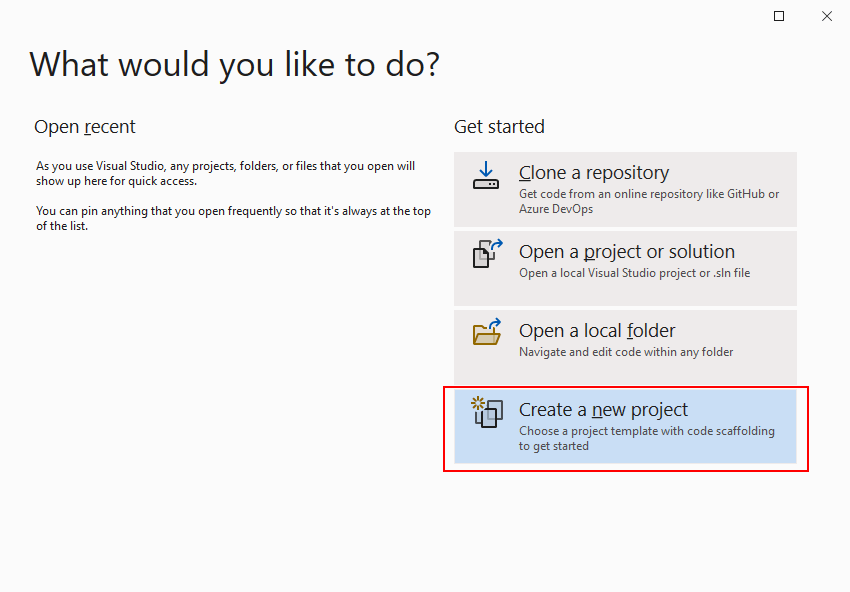
После загрузки запустим программу установщика. В открывшемся окне нам будет предложено выбрать те компоненты, которые мы хотим установить вместе Visual Studio. Стоит отметить, что Visual Studio - очень функциональная среда разработки и позволяет разрабатывать приложения с помощью множества языков и платформ. В нашем случае нам будет интересовать прежде всего C# и .NET.

Чтобы добавить в Visual Studio поддержку проектов для C# и .NET 7, в программе установки среди рабочих нагрузок можно выбрать только пункт **ASP.NET и разработка веб-приложений**. Можно выбрать и больше опций или вообще все опции, однако стоит учитывать свободный размер на жестком диске - чем больше опций будет выбрано, соответственно тем больше места на диске будет занято.

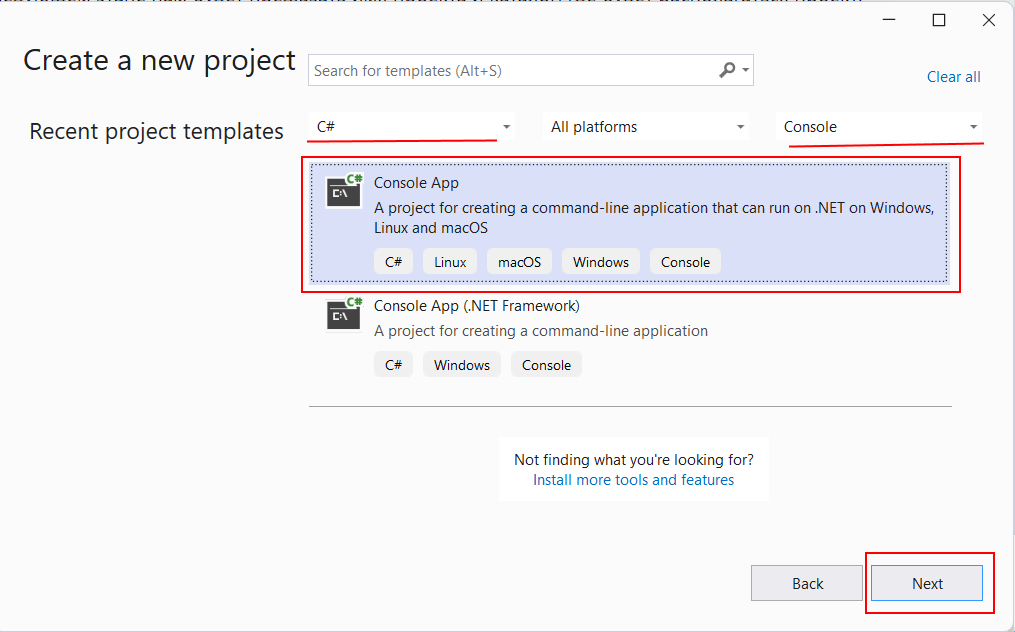


И при инсталляции Visual Studio на ваш компьютер будут установлены все необходимые инструменты для разработки программ, в том числе фреймворк .NET 7.

После завершения установки создадим первую программу. Она будет простенькой. Вначале откроем Visual Studio. На стартовом экране выберем **Create a new project** (Создать новый проект)

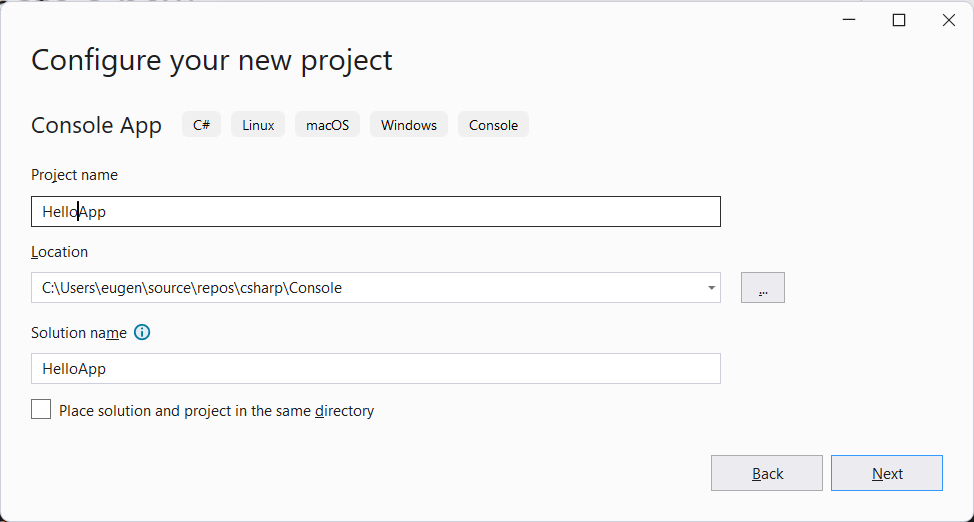


На следующем окне в качестве типа проекта выберем **Console App**, то есть мы будем создавать консольное приложение на языке C#



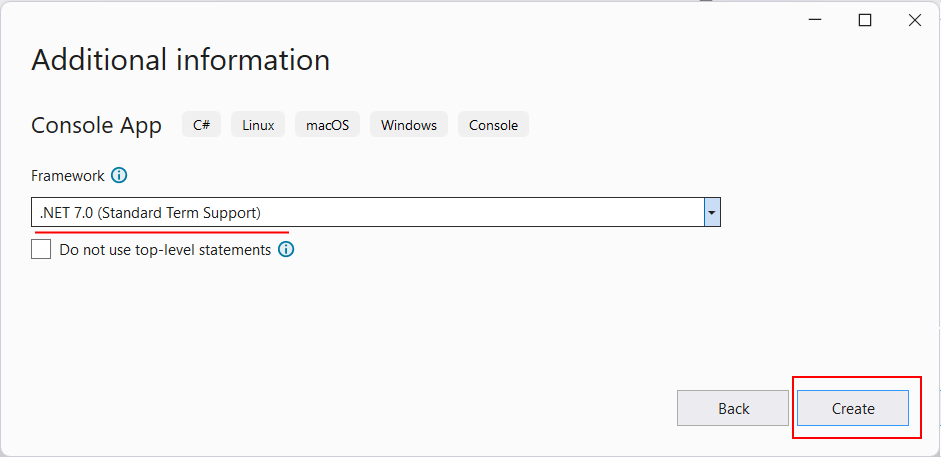
Чтобы проще было найти нужный тип проекта, в поле языков можно выбрать **C#**, а в поле типа проектов - **Console**.

Далее на следующем этапе нам будет предложено указать имя проекта и каталог, где будет располагаться проект.

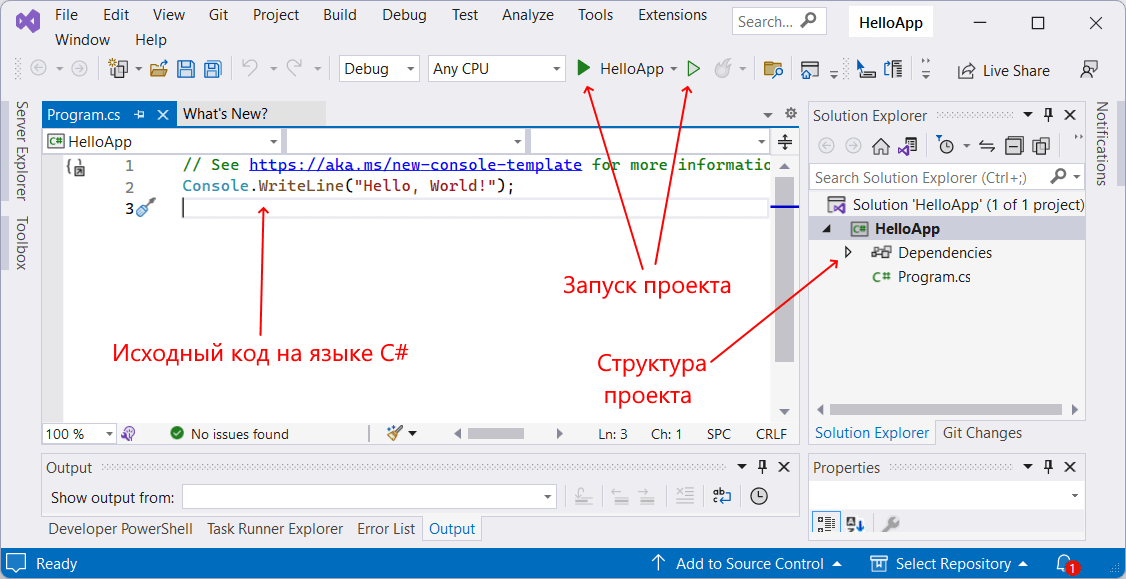


В поле **Project Name** дадим проекту какое-либо название. В моем случае это **HelloApp**.

На следующем окне Visual Studio предложит нам выбрать версию .NET, которая будет использоваться для проекта. Выберем последнюю на данный момент верси. - .NET 7.0:



Нажмен на кнопку Create (Создать) для создания проекта, и после этого Visual Studio создаст и откроет нам проект:



В большом поле в центре, которое по сути представляет текстовый редактор, находится сгенерированный по умолчанию код C#. Впоследствии мы изменим его на свой.

Справа находится окно Solution Explorer, в котором можно увидеть структуру нашего проекта. В данном случае у нас сгенерированная по умолчанию структура: узел **Dependencies** - это узел содержит сборки dll, которые добавлены в проект по умолчанию. Эти сборки как раз содержат классы библиотеки .NET, которые будет использовать C#. Однако не всегда все сборки нужны. Ненужные потом можно удалить, в то же время если понадобится добавить какую-нибудь нужную библиотеку, то именно в этот узел она будет добавляться.

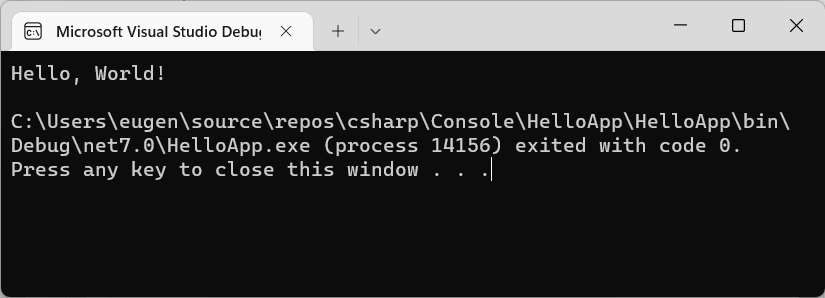
Далее идет непосредственно сам файл кода программы **Program.cs**, который по умолчанию открыт в центральном окне и который имеет всего две строки:

|  |  |
| --- | --- |
| 1  2 | // See <https://aka.ms/new-console-template> for more information  Console.WriteLine("Hello, World!"); |

Первая строка предваряется символами // и представляет комментарии - пояснения к коду.

Вторая строка собственно представляет собой код программы: Console.WriteLine("Hello World!");. Эта строка выводит на консоль строку "Hello World!".

Несмотря на то, что программа содержит только одну строку кода, это уже некоторая программа, которую мы можем запустить. Запустить проект мы можем с помощью клавиши F5 или с панели инструментов, нажав на зеленую стрелку. И если вы все сделали правильно, то при запуске приложения на консоль будет выведена строка "Hello World!".



Теперь изменим весь этот код на следующий:

|  |  |
| --- | --- |
| 1  2  3 | Console.Write("Введите свое имя: ");  var name = Console.ReadLine(); // вводим имя  Console.WriteLine($"Привет {name}"); // выводим имя на консоль |

По сравнению с автоматически сгенерированным кодом я внес несколько изменений. Теперь первой строкой выводится приглашение к вводу.

|  |  |
| --- | --- |
| 1 | Console.Write("Введите свое имя: "); |

Метод **Console.Write()** выводит на консоль некоторую строку. В данном случае это строка "Введите свое имя: ".

На второй строке определяется строковая переменная name, в которую пользователь вводит информацию с консоли:

|  |  |
| --- | --- |
| 1 | var name = Console.ReadLine(); |

Ключевое слово **var** указывает на определение переменной. В данном случае переменная называется name. И ей присваивается результат метода **Console.ReadLine()**, который позволяет считать с консоли введенную строку. То есть мы введем в консоли строку (точнее имя), и эта строка окажется в переменой name.

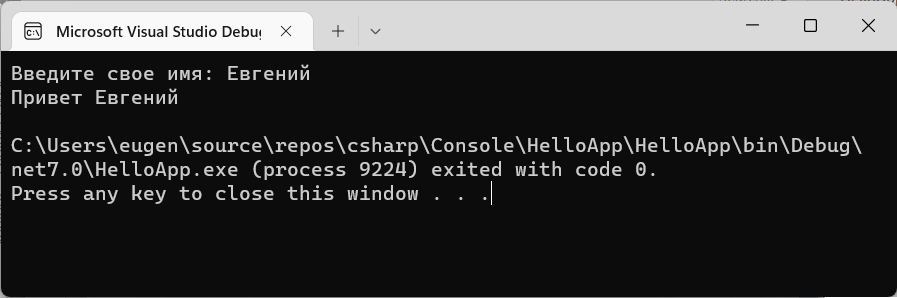
Затем введенное имя выводится на консоль:

|  |  |
| --- | --- |
| 1 | Console.WriteLine($"Привет {name}"); |

Чтобы ввести значение переменной name внутрь выводимой на консоль строки, применяются фигурные скобки {}. То есть при выводе строки на консоль выражение {name} будет заменяться на значение переменной name - введенное имя.

Однако чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара $.

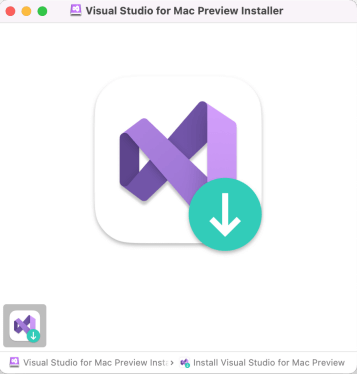
Теперь протестируем проект, запустив его на выполнение, также нажав на F5 или зеленую стрелочку.



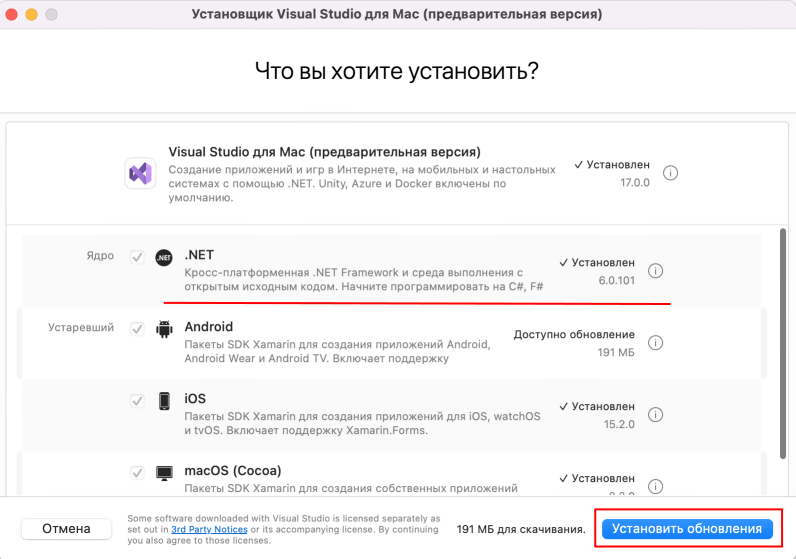
Скомпилированное приложение можно найти в папке проекта в каталоге **bin\Debug\net7.0**. Оно будет называться по имени проекта и иметь расширение exe. И затем этот файл можно будет запускать без Visual Studio, а также переносить его на другие компьютеры, где установлен .NET 7.

Фреймворк .NET является кроссплатформенным, соответственно создать приложения на C# можно не только на Windows, но и на других поддерживаемых операционных системах, например, на MacOS. Для работы с C# на MacOS можно использовать как .NET CLI, писать код программы в любом текстовом редакторе и компилировать в терминале. Но также для работы в MacOS можно использовать среду разработки **Visual Studio for Mac**. Для этого нам надо сначала загрузить установщик данной среды разработки со страницы <https://visualstudio.microsoft.com/ru/vs/mac/>. Для работы с .NET 7 и C# нам потребуется именно версия **Visual Studio for Mac 2022**

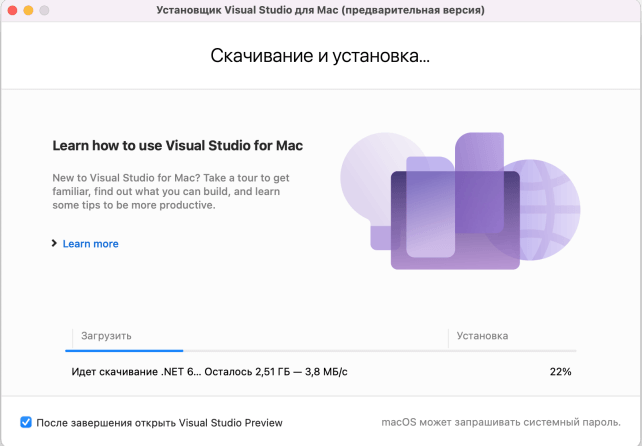
После загрузки запустим программу установки и нажмем на кнопку установки:



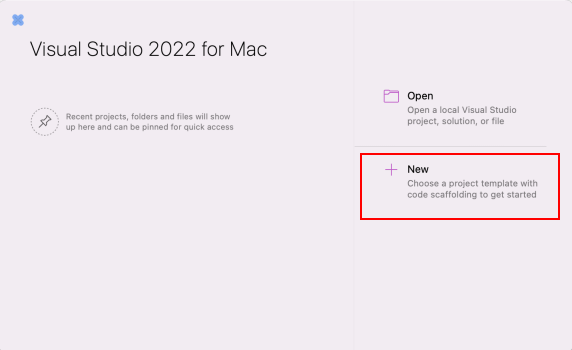
Далее нам откроется программа установки, где нам надо указать устанавливаемые компоненты. Если Visual Studio или отдельные компоненты уже были установлены, то отобразиться список компонентов, которые надо обновить:



Нажмем на кнопку установки, и начнется процесс установки:

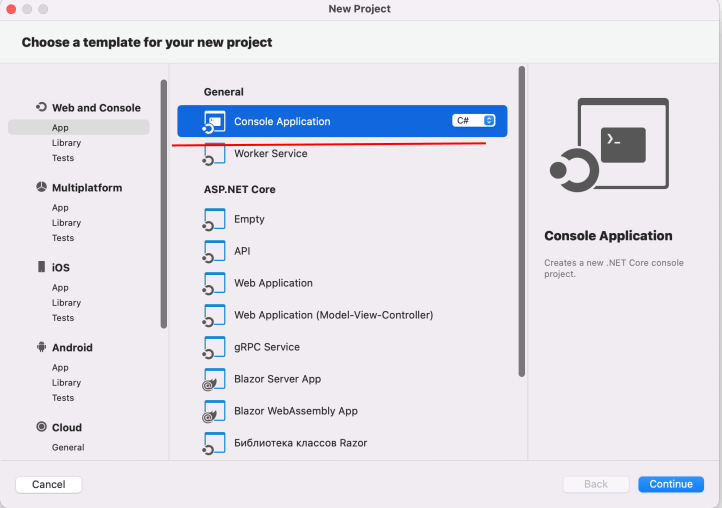


После завершения установки откроем Visual Studio for Mac:

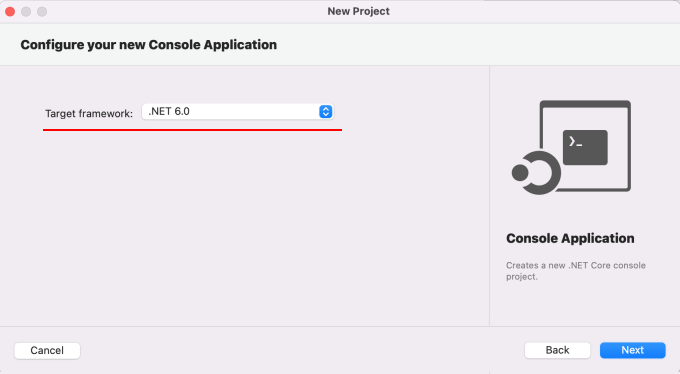


Для создания нового проекта нажмем на кнопку **New**

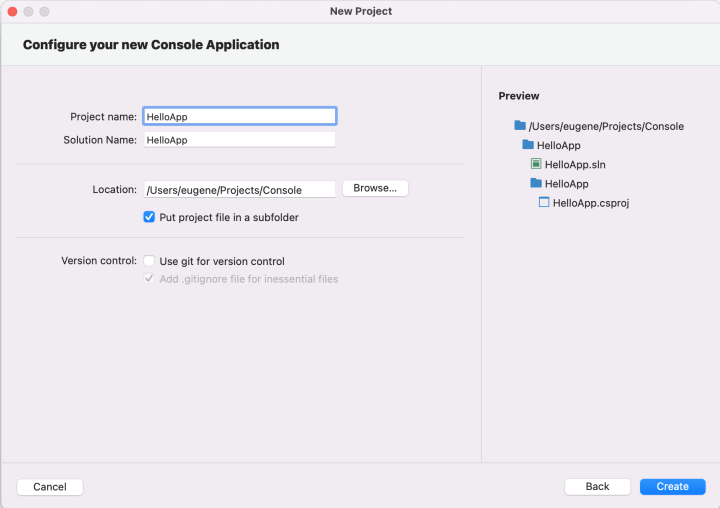
Далее нам откроется окно выбора шаблона проекта. В данном случае выберем шаблон **Console Application**



Далее в следующем окне укажем версию фреймворка .NET:



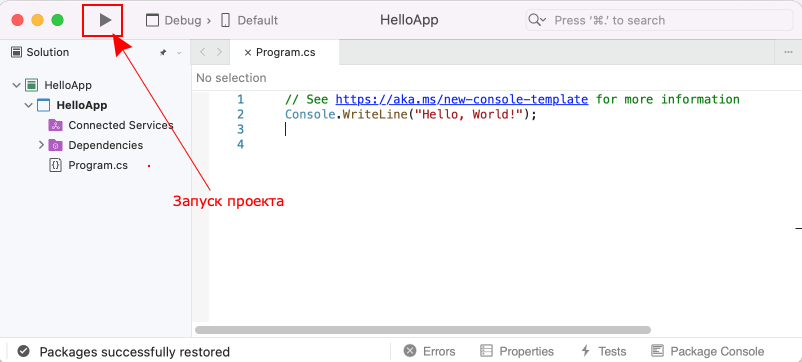
Далее в следующем окне укажем имя проекта и его расположение на жестком диске. Пусть проект называется HelloApp:



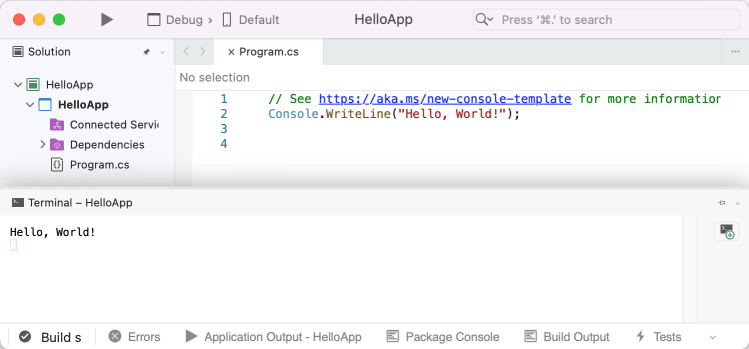
После этого Visual Studio создаст новый проект. Как и в версии для Windows, проект имеет один файл Program.cs. Он содержит простейший код, которые выводит сообщение на консоль:

|  |  |
| --- | --- |
| 1 | Console.WriteLine("Hello, World!"); |

Запустим проект, нажав на черную стрелку над структурой проекта:



И после запуска внизу в Visiual Studio for MacOS отобразится выше указанная строка:



В остальных аспектах разработка приложений на .NET и C# на MacOS будет аналогична разработке в Windows.

.NET является кроссплатформенным фреймворком и подддерживается в том числе различными дистрибутивами на базе Linux. В данном случае рассмотрим создание первой программы на ОС Ubuntu.

### Установка .NET

Для начала работы с C# и .NET на Linux сначала надо установить **.NET SDK**. Установка конкретной версии .NET зависит от конкретной версии Ubuntu. Например, на момент написания данной статьи только версия .NET 6 SDK включена в пакетный менеджер Ubuntu. И чтобы установить .NET 6, в терминале можно ввести команду

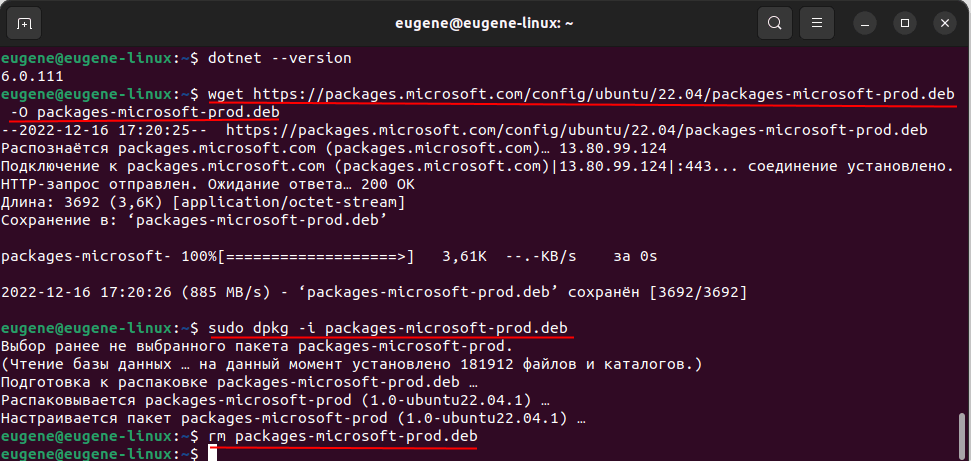
sudo apt-get update && sudo apt-get install -y dotnet6

После установки мы можем проверить версию .NET SDK с помощью команды:

dotnet --version

Но текущей версией .NET SDK является версия 7. Но на данный момент отстуствует соответствующий канал на эту версию в пакетном менеджере. В будущем ситуация может изменить, но сейчас надо выполнить чуть больше команд. Поэтому сначала необходимо добавить репозиторий пакетов Microsoft с помощью последовательного выполнения следующих команд

wget <https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb> -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb



Обратите внимание на циферки в первой команде - "22.04". В случае если используется другая версия Ubuntu, то вместо них указывается соответствующая версия. Соответственно для Ubuntu 22.10 команда для установки репозитория будет выглядеть так

wget <https://packages.microsoft.com/config/ubuntu/22.10/packages-microsoft-prod.deb> -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb

Затем для установки 7-й версии .NET SDK выполним команду

sudo apt-get update && sudo apt-get install -y dotnet-sdk-7.0

Для проверки установленных .NET SDK можно выполнить следующую команду

dotnet --list-sdks

### Первый проект

Создадим первый проект. Для этого будем использовать набор консольных утилит .NET CLI. И вначале создадим каталог, где будет располагаться наш проект. Например, в моем случае это каталог /home/eugene/metanit/dotnet/helloapp. Откроем терминал и перейдем к этому каталогу с помощью команды **cd**.

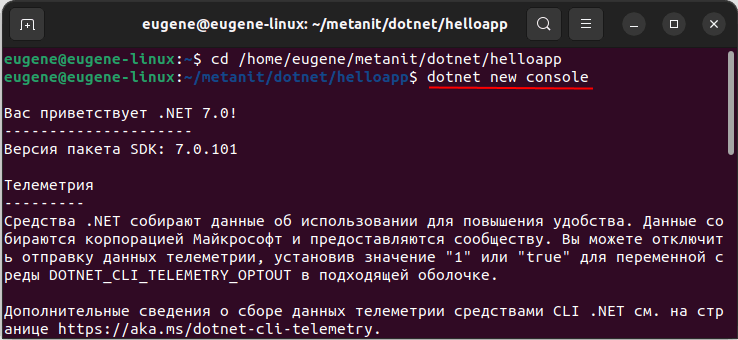
Затем перейдем к этой папке командой

cd /home/eugene/metanit/dotnet/helloapp

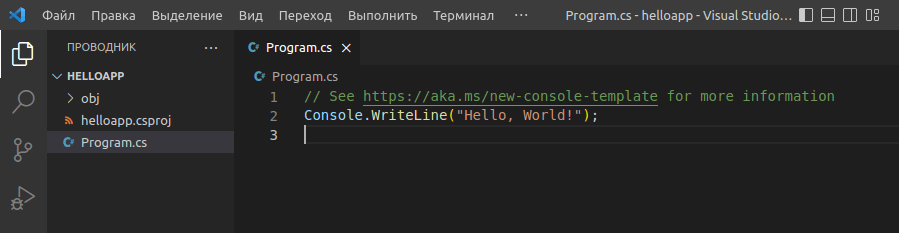
Затем для создания проекта в этом каталоге выполним следующую команду:

dotnet new console

В данном случае мы указываем, что создаваемый проект будет представлять консольное приложение.



После выполнения этой команды в папке helloapp будет создан проект с минимальным набором стандартных файлов и папок.



В частности, мы можем найти в папке проекта файл **helloapp.csproj**. Это главный файл проекта, который определяет его конфигурацию. Мы можем открыть его в любом текстовом редакторе, просмотреть и при необходимости изменить.

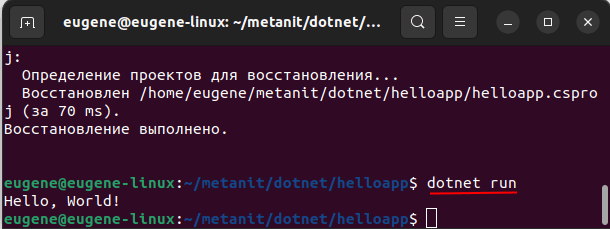
И, кроме того, по умолчанию создается главный файл программы **Program.cs** со следующим содержимым:

|  |  |
| --- | --- |
| 1  2 | // See <https://aka.ms/new-console-template> for more information  Console.WriteLine("Hello, World!"); |

Соответственно, если нам надо внести изменения в исходный код программы, то мы будем изменять содержимое именно этого файла.

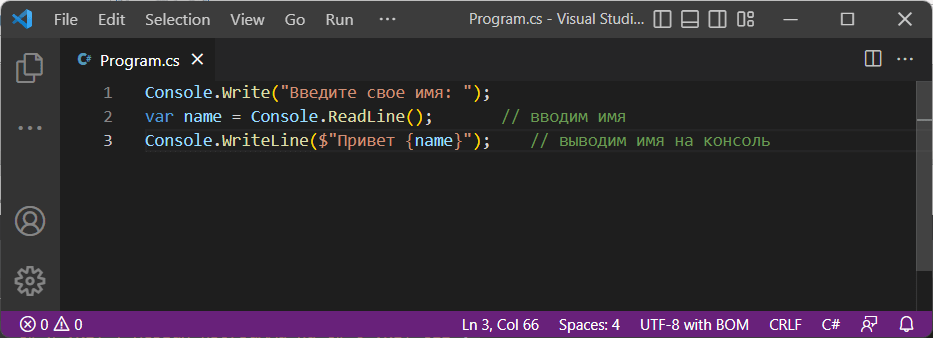
В принципе этот минимальный проект уже можно запускать. Для запуска проекта введем в командной строке следующую команду:

dotnet run



Теперь изменим весь код программы. Для этого откроем файл **Program.cs** в каком-нибудь текстовом редакторе и изменим его код на следующий:

|  |  |
| --- | --- |
| 1  2  3 | Console.Write("Введите свое имя: ");  var name = Console.ReadLine(); // вводим имя  Console.WriteLine($"Привет {name}"); // выводим имя на консоль |



По сравнению с автоматически сгенерированным кодом я внес несколько изменений. Теперь первой строкой выводится приглашение к вводу.

|  |  |
| --- | --- |
| 1 | Console.Write("Введите свое имя: "); |

Метод **Console.Write()** выводит на консоль некоторую строка. В данном случае это строка "Введите свое имя: ".

На второй строке определяется строковая переменная name, в которую пользователь вводит информацию с консоли:

|  |  |
| --- | --- |
| 1 | var name = Console.ReadLine(); |

Ключевое слово **var** указывает на определение переменной. В данном случае переменная называется name. И ей присваивается результат метода **Console.ReadLine()**, который позволяет считать с консоли введенную строку. То есть мы введем в консоли строку (точнее имя), и эта строка окажется в переменой name.

Затем введенное имя выводится на консоль:

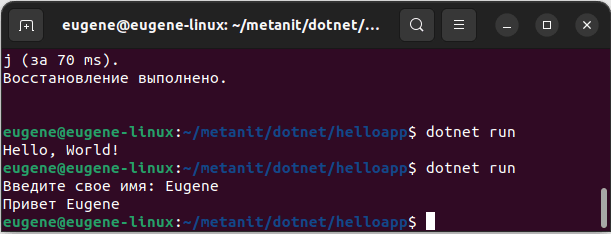
|  |  |
| --- | --- |
| 1 | Console.WriteLine($"Привет {name}"); |

Чтобы ввести значение переменной name внутрь выводимой на консоль строки, применяются фигурные скобки {}. То есть при выводе строки на консоль выражение {name} будет заменяться на значение переменной name - введенное имя.

Однако чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара $.

Теперь протестируем проект, запустив его на выполнение с помощью команды

dotnet run



.NET является кроссплатформенным фреймворком и подддерживается в том числе различными дистрибутивами на базе Linux. В данном случае рассмотрим создание первой программы на ОС Ubuntu.

### Установка .NET

Для начала работы с C# и .NET на Linux сначала надо установить **.NET SDK**. Установка конкретной версии .NET зависит от конкретной версии Ubuntu. Например, на момент написания данной статьи только версия .NET 6 SDK включена в пакетный менеджер Ubuntu. И чтобы установить .NET 6, в терминале можно ввести команду

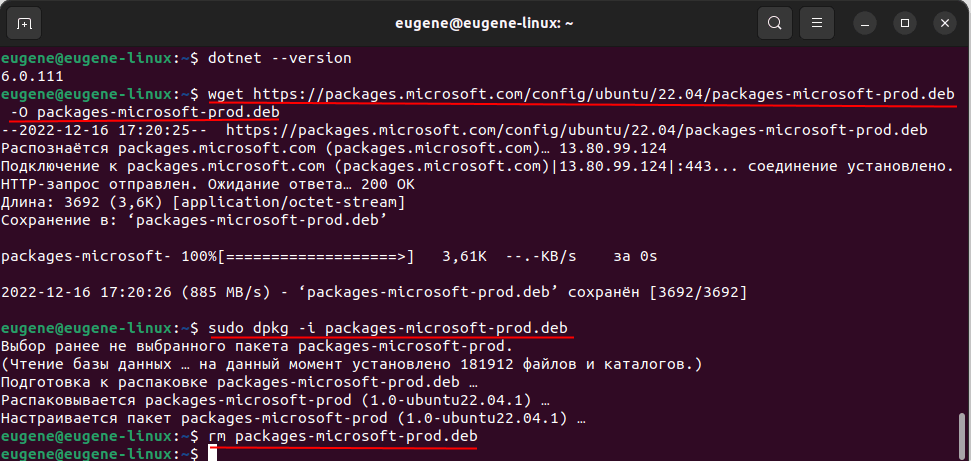
sudo apt-get update && sudo apt-get install -y dotnet6

После установки мы можем проверить версию .NET SDK с помощью команды:

dotnet --version

Но текущей версией .NET SDK является версия 7. Но на данный момент отстуствует соответствующий канал на эту версию в пакетном менеджере. В будущем ситуация может изменить, но сейчас надо выполнить чуть больше команд. Поэтому сначала необходимо добавить репозиторий пакетов Microsoft с помощью последовательного выполнения следующих команд

wget <https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.deb> -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb



Обратите внимание на циферки в первой команде - "22.04". В случае если используется другая версия Ubuntu, то вместо них указывается соответствующая версия. Соответственно для Ubuntu 22.10 команда для установки репозитория будет выглядеть так

wget <https://packages.microsoft.com/config/ubuntu/22.10/packages-microsoft-prod.deb> -O packages-microsoft-prod.deb  
sudo dpkg -i packages-microsoft-prod.deb  
rm packages-microsoft-prod.deb

Затем для установки 7-й версии .NET SDK выполним команду

sudo apt-get update && sudo apt-get install -y dotnet-sdk-7.0

Для проверки установленных .NET SDK можно выполнить следующую команду

dotnet --list-sdks

### Первый проект

Создадим первый проект. Для этого будем использовать набор консольных утилит .NET CLI. И вначале создадим каталог, где будет располагаться наш проект. Например, в моем случае это каталог /home/eugene/metanit/dotnet/helloapp. Откроем терминал и перейдем к этому каталогу с помощью команды **cd**.

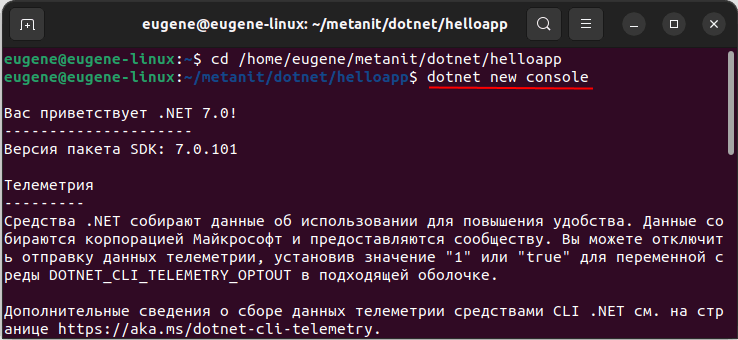
Затем перейдем к этой папке командой

cd /home/eugene/metanit/dotnet/helloapp

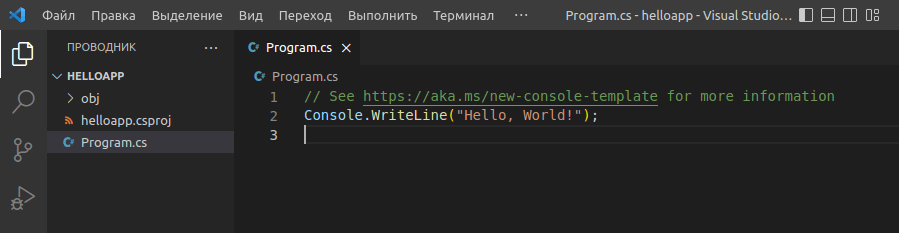
Затем для создания проекта в этом каталоге выполним следующую команду:

dotnet new console

В данном случае мы указываем, что создаваемый проект будет представлять консольное приложение.



После выполнения этой команды в папке helloapp будет создан проект с минимальным набором стандартных файлов и папок.



В частности, мы можем найти в папке проекта файл **helloapp.csproj**. Это главный файл проекта, который определяет его конфигурацию. Мы можем открыть его в любом текстовом редакторе, просмотреть и при необходимости изменить.

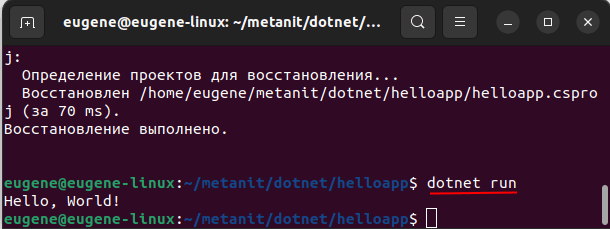
И, кроме того, по умолчанию создается главный файл программы **Program.cs** со следующим содержимым:

|  |  |
| --- | --- |
| 1  2 | // See <https://aka.ms/new-console-template> for more information  Console.WriteLine("Hello, World!"); |

Соответственно, если нам надо внести изменения в исходный код программы, то мы будем изменять содержимое именно этого файла.

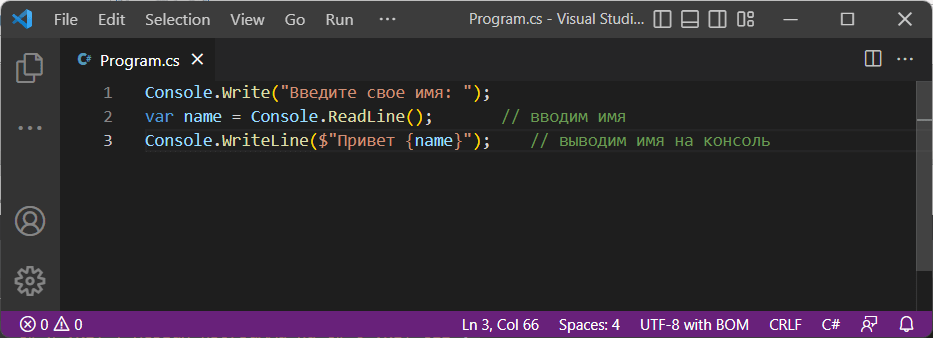
В принципе этот минимальный проект уже можно запускать. Для запуска проекта введем в командной строке следующую команду:

dotnet run



Теперь изменим весь код программы. Для этого откроем файл **Program.cs** в каком-нибудь текстовом редакторе и изменим его код на следующий:

|  |  |
| --- | --- |
| 1  2  3 | Console.Write("Введите свое имя: ");  var name = Console.ReadLine(); // вводим имя  Console.WriteLine($"Привет {name}"); // выводим имя на консоль |



По сравнению с автоматически сгенерированным кодом я внес несколько изменений. Теперь первой строкой выводится приглашение к вводу.

|  |  |
| --- | --- |
| 1 | Console.Write("Введите свое имя: "); |

Метод **Console.Write()** выводит на консоль некоторую строка. В данном случае это строка "Введите свое имя: ".

На второй строке определяется строковая переменная name, в которую пользователь вводит информацию с консоли:

|  |  |
| --- | --- |
| 1 | var name = Console.ReadLine(); |

Ключевое слово **var** указывает на определение переменной. В данном случае переменная называется name. И ей присваивается результат метода **Console.ReadLine()**, который позволяет считать с консоли введенную строку. То есть мы введем в консоли строку (точнее имя), и эта строка окажется в переменой name.

Затем введенное имя выводится на консоль:

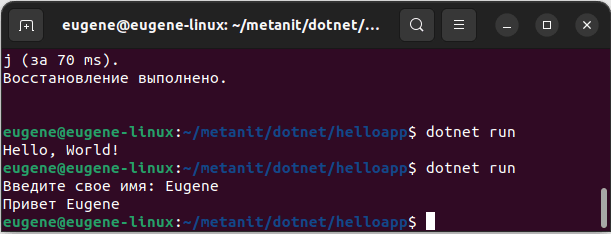
|  |  |
| --- | --- |
| 1 | Console.WriteLine($"Привет {name}"); |

Чтобы ввести значение переменной name внутрь выводимой на консоль строки, применяются фигурные скобки {}. То есть при выводе строки на консоль выражение {name} будет заменяться на значение переменной name - введенное имя.

Однако чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара $.

Теперь протестируем проект, запустив его на выполнение с помощью команды

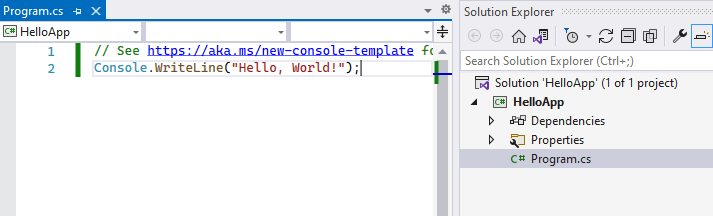
dotnet run



### Выполнение программы

Весь код программы на языке C# помещается в файлы с расширением **.cs**. По умолчанию в проекте, который создается в Visual Studio (а также при использовании .NET CLI) уже есть один файл с кодом C# - файл **Program.cs** со следующим содержимым:

|  |  |
| --- | --- |
| 1  2 | // See <https://aka.ms/new-console-template> for more information  Console.WriteLine("Hello, World!"); |



Именно код файла **Program.cs** выполняется по умолчанию, если мы запустим проект на выполнение. Но при необходимости мы также можем добавлять другие файлы с кодом C#.

### Инструкции

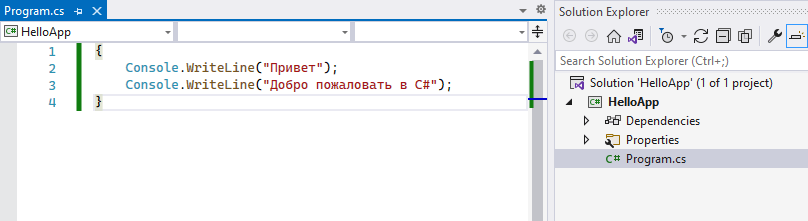
Базовым строительным блоком программы являются **инструкции** (statement). Инструкция представляет некоторое действие, например, арифметическую операцию, вызов метода, объявление переменной и присвоение ей значения. В конце каждой инструкции в C# ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции. Например, в проекте консольного приложения, который создается по умолчанию, есть такая строка:

|  |  |
| --- | --- |
| 1 | Console.WriteLine("Hello, World!"); |

Данная строка представляет вызов метода Console.WriteLine, который выводит на консоль строку. В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может объединяться в блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками. Например, изменим код файла **Program.cs** на следующий:

|  |  |
| --- | --- |
| 1  2  3  4 | {  Console.WriteLine("Привет");  Console.WriteLine("Добро пожаловать в C#");  } |



Здесь блок кода содержит две инструкции. И при выполении этого кода, консоль выведет две строки

Привет  
Добро пожаловать в C#

В данном блоке кода две инструкции, которые выводят на консоль определенную строку.

Одни блоки кода могут содержать другие блоки:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {  Console.WriteLine("Первый блок");  {  Console.WriteLine("Второй блок");  }  } |

### Регистрозависимость

C# является регистрозависимым языком. Это значит, что в зависимости от регистра символов какие-то определенные названия могут представлять разные классы, методы, переменные и т.д. Например, для вывода на консоль используется метод **WriteLine** - его имя начинается именно с большой буквы: "WriteLine". Если мы вместо "Console.WriteLine" напишем "Console.writeline", то программа не скомпилируется, так как данный метод обязательно должен называться "WriteLine", а не "writeline" или "WRITELINE" или как-то иначе.

### Комментарии

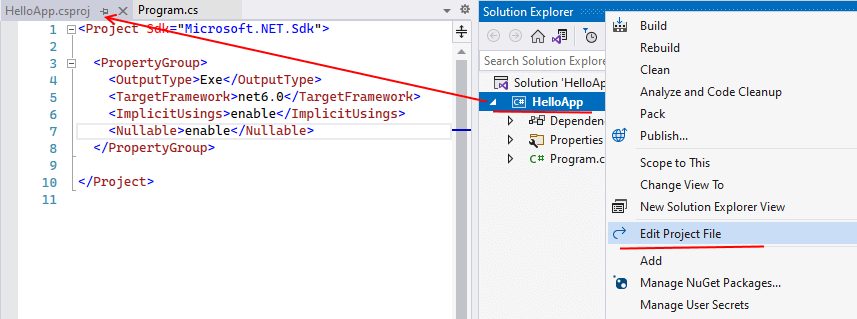
Важной частью программного кода являются комментарии. Они не являются собственно частью программы, при компиляции они игнорируются. Тем не менее комментарии делают код программы более понятным, помогая понять те или иные его части.

есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //. А многострочный комментарий заключается между символами /\* текст комментария \*/. Он может размещаться на нескольких строках. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\*  первая программа на C#,  которая выводит приветствие на консоль  \*/  Console.WriteLine("Привет"); // Выводим строку "Привет"  Console.WriteLine("Добро пожаловать в C#"); // Выводим строку "Добро пожаловать в C#" |

### Файл проекта

В каждом проекте проекте C# есть файл, который отвечает за общую конфгурацию проекта. По умолчанию этот файл называется **Название\_проекта.csproj**. Итак, откроем данный файл. Для этого либо двойным кликом левой кнопкой мыши нажмем на название проекта, либо нажмем на название проекта правой кнопкой мыши и в появившемся меню выберем пункт **Edit Project File**



После этого Visual Studio откроет нам файл проекта, который будет выглядеть следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <Project Sdk="Microsoft.NET.Sdk">    <PropertyGroup>  <OutputType>Exe</OutputType>  <TargetFramework>net6.0</TargetFramework>  <ImplicitUsings>enable</ImplicitUsings>  <Nullable>enable</Nullable>  </PropertyGroup>    </Project> |

Этот файл в виде кода xml определяет конфгурацию проекта и он может содержать множество элементов. Остановлюсь только на двух основных:

* **OutputType**: определяет выходной тип проекта. Это может быть выполняемое приложение в виде файла с расширением **exe**, которое запускается по нажатию. И также это может быть файл с расширением **.dll** - некоторый набор функциональностей, который используется другими проектами. По умолчанию здесь установлено значение "Exe", что значит, что мы создаем исполняемое приложение.
* **TargetFramework**: определяет применяемую для компиляции версию фреймворка .NET. В данном случае это значение "net6.0", то есть применяется .NET 6.0.

На самых ранних этапах этот файл может не понадобиться, однако впоследствии может потребоваться внести некоторые изменения в конфигурацию, и тогда может возникнуть потребность в обращении к этому файлу.