

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

ЛЕКЦІЯ №1

“В своей книге Герберт Шилдт освещает все, что требуется знать для применения C# 4.0 на практике”.

Майкл Ховард, корпорация Microsoft

ПОЛНОЕ
РУКОВОДСТВО

C# 4.0

Книга содержит:

- Полное описание средств языка C#
- Подробное рассмотрение новых средств в версии C# 4.0, в том числе PLINQ, библиотека TPL, именованные и необязательные аргументы, динамический тип данных и многое другое
- Сотни простых и понятных примеров программ с комментариями

БЕСПЛАТНЫЙ
КОД
В ИНТЕРНЕТЕ

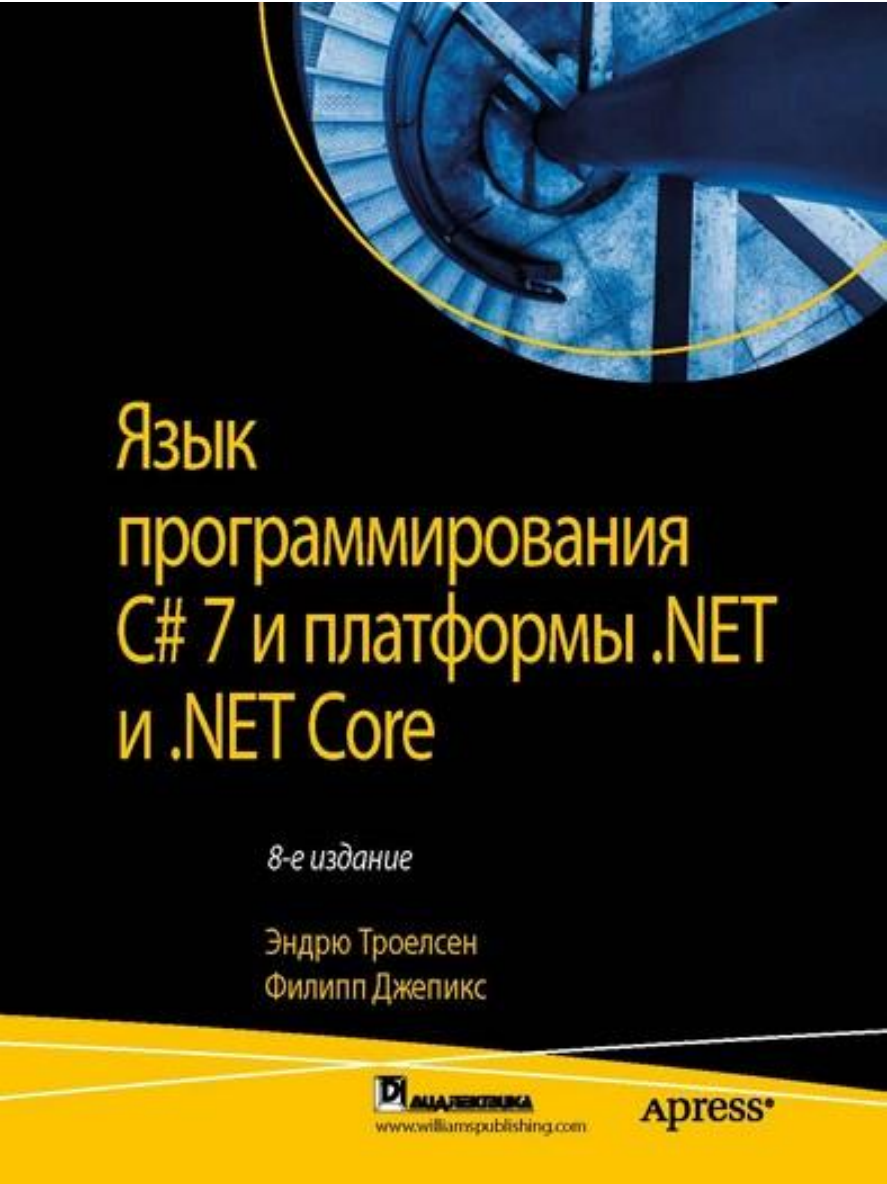
БОНУС!

WWW.OSBORNE.COM

Полностью исправленное
и обновленное издание классического
руководства по C# 4.0

Герберт Шилдт

Автор лучших книг по программированию,
проданных миллионными тиражами по всему миру!



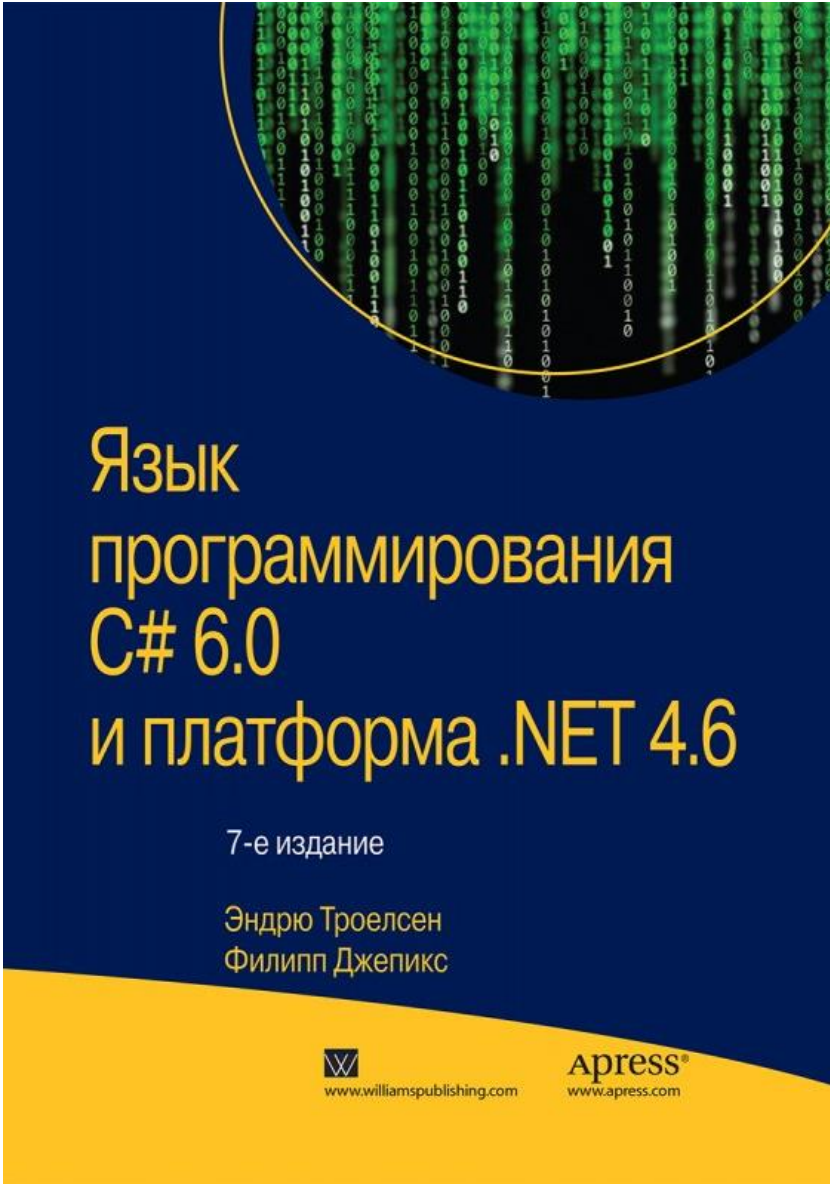
Язык программирования C# 7 и платформы .NET и .NET Core

8-е издание

Эндрю Троелсен
Филипп Джепикс

 **АПАРЕССА**
www.williamspublishing.com

Apress®



Язык программирования C# 6.0 и платформа .NET 4.6

7-е издание

Эндрю Троелсен
Филипп Джепикс


www.williamspublishing.com

Apress®
www.apress.com

METANIT.COM

Сайт о программировании

Professor Web

**Уроки по C# и
платформе .NET
Framework**



About

Documentation

- Reference
- Book
- Videos
- External Links

Downloads

Community

This book is available in [English](#).

Full translation available in

[български език](#),
[Español](#),
[Français](#),
[Ελληνικά](#),
[日本語](#),
[한국어](#),
[Nederlands](#),
[Русский](#),
[Slovenščina](#),
[Tagalog](#),
[Українська](#),
[简体中文](#),

Partial translations available in

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).

. Введение

- 1 О контроле версий
- 2 Краткая история Git
- 3 Основы Git
- 4 Установка Git
- 5 Первоначальная настройка Git
- 6 Как получить помощь?
- 7 Итоги

. Основы Git

- 1 Создание Git-репозитория
- 2 Запись изменений в репозиторий
- 3 Просмотр истории коммитов
- 4 Отмена изменений
- 5 Работа с удалёнными репозиториями
- 6 Работа с метками
- 7 Полезные советы
- 8 Итоги



1st Edition (2009)

[Switch to 2nd Edition](#)

C# достаточно молодой, но в то же время он 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# 8.0, которая вышла в сентябре 2019 года вместе с релизом .NET Core 3.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java.

Роль платформы .NET

•**Поддержка нескольких языков.** Основой платформы является общезыковая 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 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 Core 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.** Общезыковая 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 UWP, для создания более простых графических приложений - Windows Forms. Для разработки мобильных приложений - Xamarin. Для создания веб-сайтов - ASP.NET и т.д.

.NET Framework и .NET Core

Стоит отметить, что .NET долгое время развивался преимущественно 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. Поэтому следует различать .NET Framework, который предназначен преимущественно для Windows, и кроссплатформенный .NET Core. В данном руководстве речь будет идти о C# в связке с .NET Core, поскольку это актуальная платформа.

.NET FRAMEWORK

Platform for .NET applications on Windows

Distributed with Windows

.NET CORE

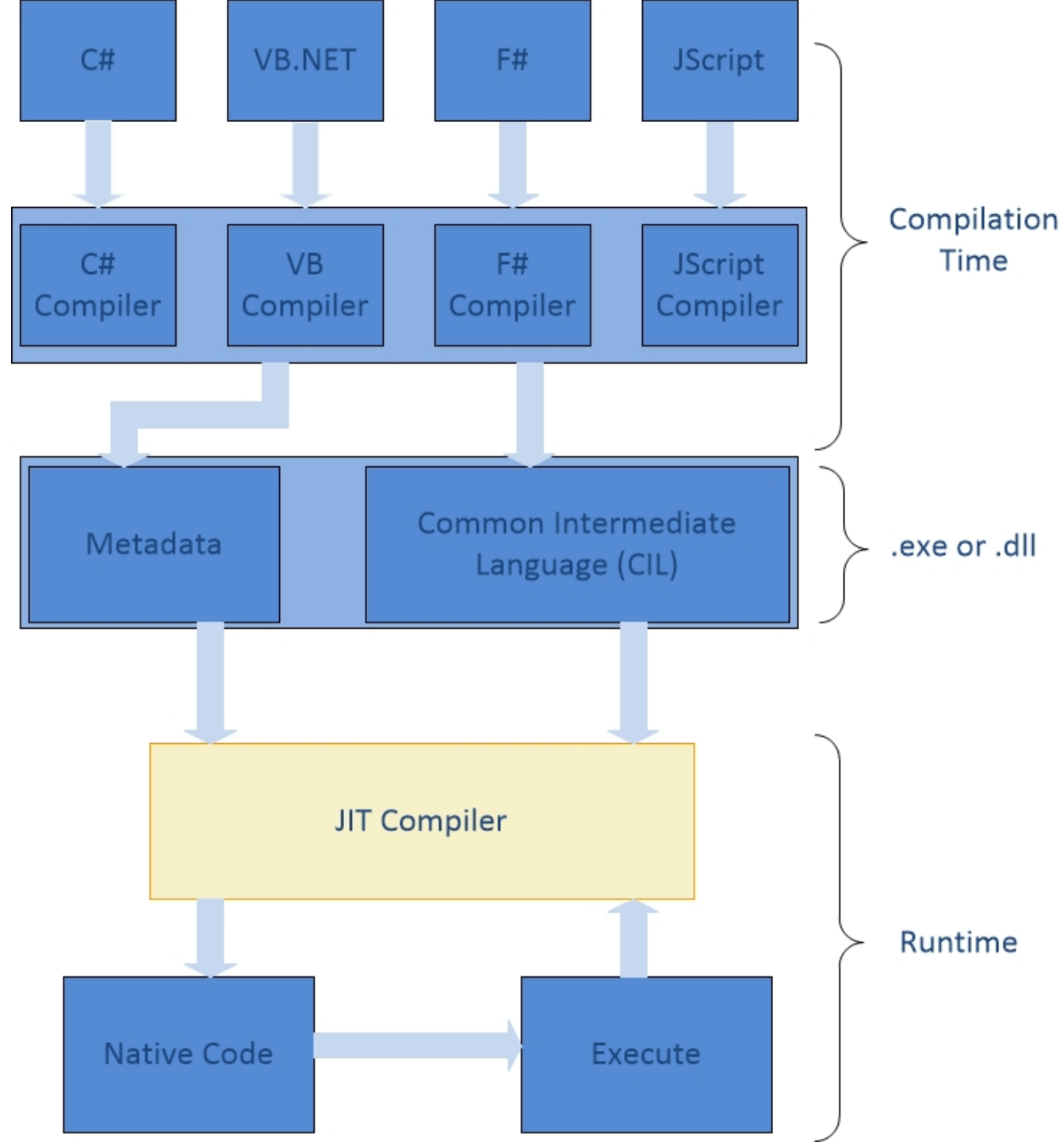
Cross-platform and open source framework optimized for modern app needs and developer workflows

Distributed with app

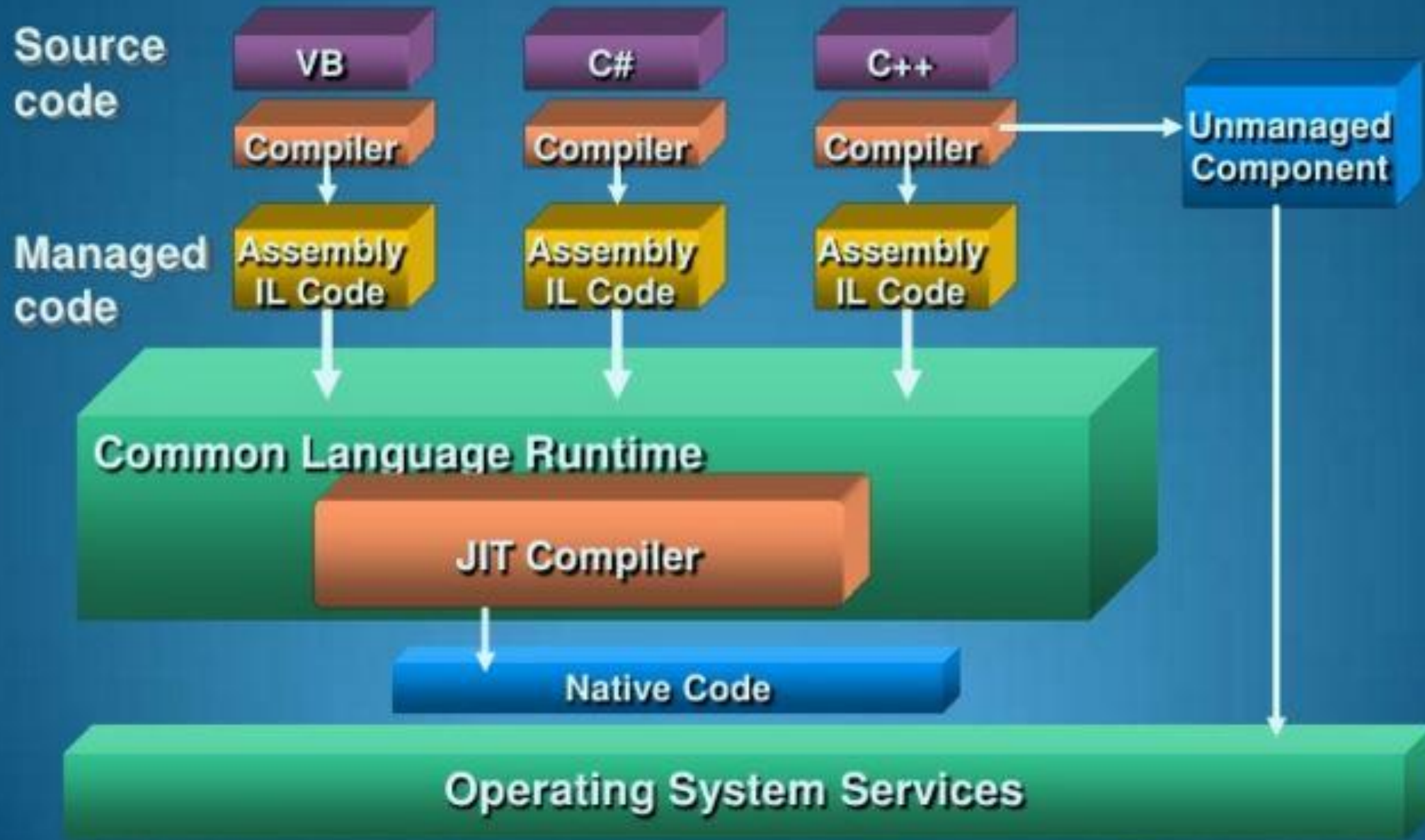
XAMARIN

Cross-platform and open source Mono-based runtime for iOS, OS X, and Android devices

Distributed with app



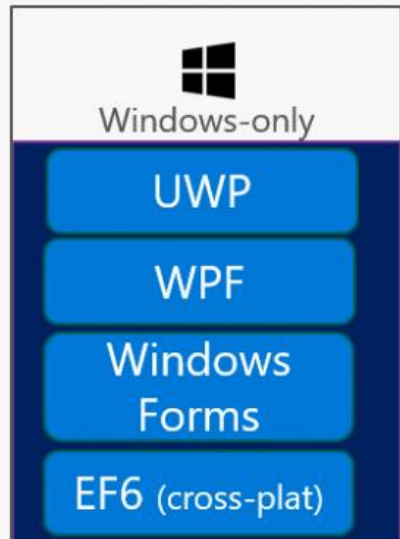
CLR Execution Model



Modernize Desktop Apps with .NET Core 3

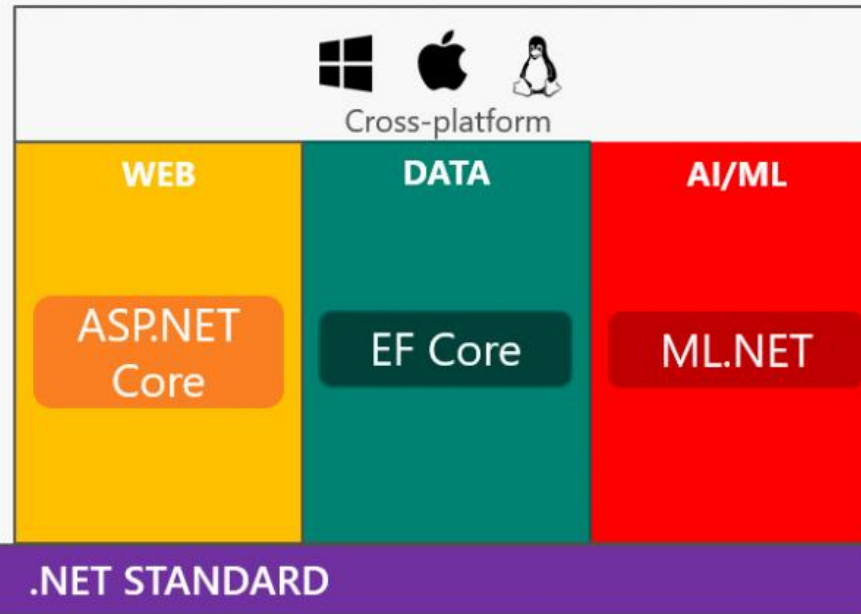


Desktop Packs



+

.NET Core 3



- XAML Islands - WinForms & WPF apps can host UWP controls
- Full access to Windows 10 APIs
- Side-by-side support & self contained exes
- Desktop pack to enable porting existing apps to .NET Core

Начало работы. Visual Studio



Версия: 16.2

[Заметки о выпуске](#)

Visual Studio 2019

Полнофункциональная интегрированная среда разработки (IDE) для приложений Android, iOS, Windows, веб- и облачных приложений

Community

Эффективная среда IDE предоставляется бесплатно для студентов, участников создания открытого исходного кода и отдельных пользователей

Загружается
бесплатно ↓

Скачать
предварительную
версию >

Professional

Профессиональная интегрированная среда разработки, оптимальная для небольших команд

Бесплатная пробная
версия ↓

Скачать
предварительную
версию >

Enterprise

Масштабируемое, комплексное решение для команд любого размера

Бесплатная пробная
версия ↓

Скачать
предварительную
версию >

[Сравнение выпусков](#)
[Автономная установка](#)
[файлов](#)



[Заметки о выпуске](#)

Visual Studio Code

Быстрый, бесплатный редактор с открытым исходным кодом, адаптирующийся под ваши потребности

Загружается
бесплатно ↓

Загружая и используя Visual Studio Code, вы принимаете [условия лицензии](#) и [заявление о конфиденциальности](#).

Начало работы



Клонирование или извлечение кода

Получить код из интернет-репозитория, например, GitHub или Azure DevOps



Открыть проект или решение

Открыть локальный проект Visual Studio или SLN-файл



Открыть локальную папку

Перейти и изменить код в любой папке



Создание проекта

Выберите шаблон проекта с формированием шаблонов кода, чтобы начать работу

[Продолжить без кода →](#)

Создание проекта

Поиск шаблонов (ALT+S) 🔍

Язык ▾

Платформа ▾

Тип проекта ▾

Последние шаблоны проектов

Здесь отобразится список недавно использованных шаблонов.



Консольное приложение (.NET Core)

Проект для создания приложения командной строки, которое может выполняться в среде .NET Core в Windows, Linux и Mac OS.

C#

Linux

macOS

Windows

Консоль



Веб-приложение ASP.NET Core

Шаблоны проектов для создания приложений ASP.NET Core в Windows, Linux и macOS с использованием .NET Core или .NET Framework. Создавайте веб-приложения с использованием Razor Pages, MVC и Blazor, а также одностраничные приложения с использованием Angular, React и React + Redux. Вы также можете разрабатывать веб-API, службы gRPC и службы рабочих ролей.

C#

Windows

Linux

macOS

Веб



Приложение WPF (.NET Framework)

Клиентское приложение Windows Presentation Foundation

C#

Windows

Рабочий стол



Библиотека классов (.NET Standard)

Проект для создания библиотеки классов, предназначенной для .NET Standard.

C#

Android

iOS

Linux

macOS

Windows

Библиотека



Функции Azure

Шаблон для создания проекта функции Azure.

C#

Azure

Облако

Configure your new project

Console App (.NET Core)

C#

Linux

macOS

Windows

Console


Project name

HelloApp

Location

C:\Users\Eugene\Source\Repos\Console\

...

Solution name 

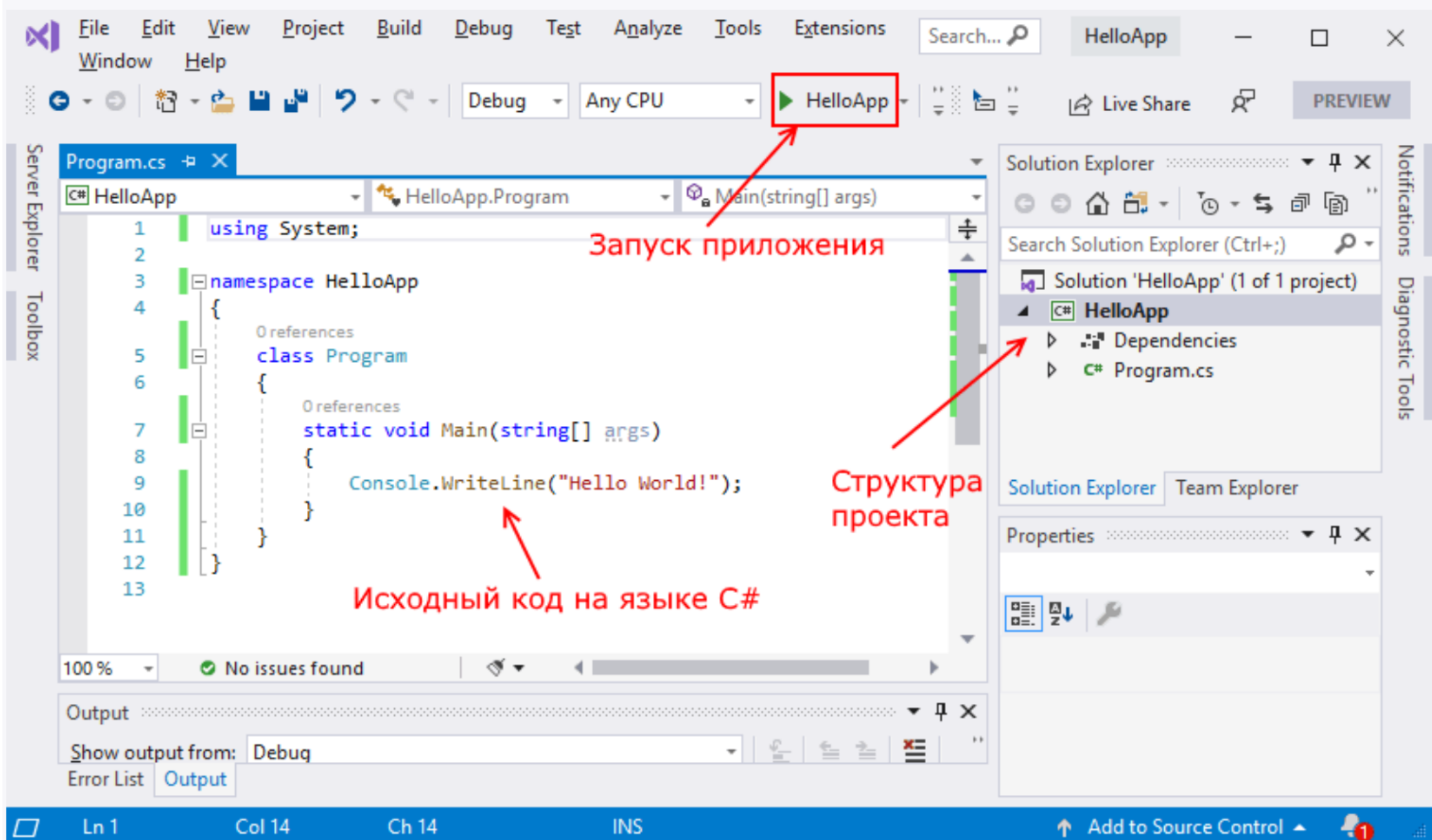
HelloApp

☐

Place solution and project in the same directory

Back

Create



```
1 using System; // подключаемое пространство имен
2
3 namespace HelloApp // объявление нового пространства имен
4 {
5     class Program // объявление нового класса
6     {
7         static void Main(string[] args) // объявление нового метода
8         {
9             Console.WriteLine("Hello World!"); // действия метода
10
11         } // конец объявления нового метода
12
13     } // конец объявления нового класса
14
15 } // конец объявления нового пространства имен
```

C:\> Microsoft Visual Studio Debug Console

Hello World!

C:\Users\Eugene\Source\Repos\Console\HelloApp\HelloApp\bin\Debug\netcoreapp3.0\HelloApp.exe (process 1144) exited with code 0.

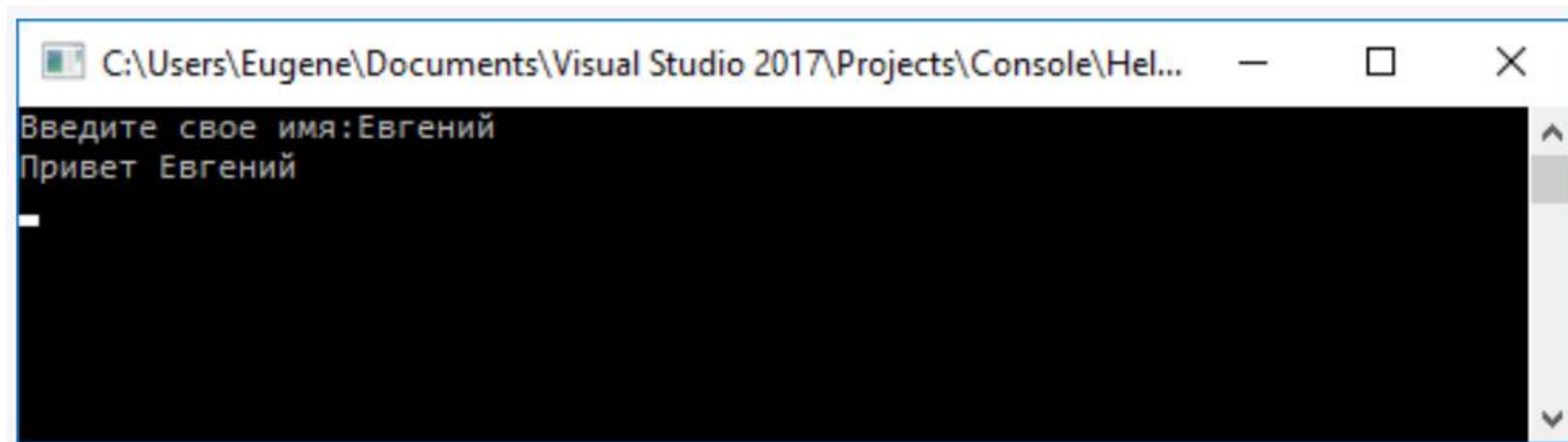
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.

Press any key to close this window . . .

■

```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.Write("Введите свое имя: ");
10            string name = Console.ReadLine();    // вводим имя
11            Console.WriteLine($"Привет {name}");  // выводим имя на консоль
12            Console.ReadKey();
13        }
14    }
15 }
```

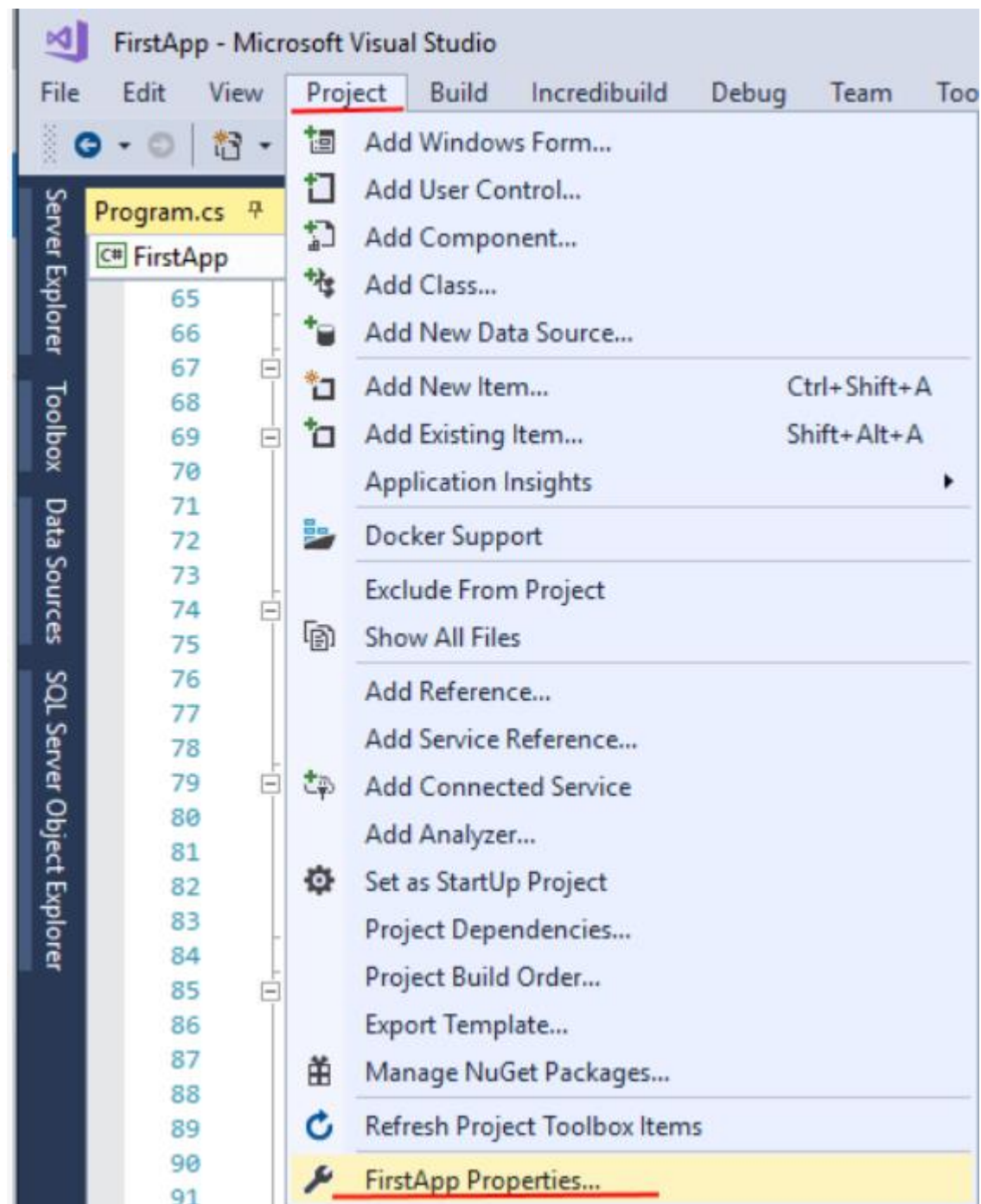
Чтобы ввести значение переменной name внутрь выводимой на консоль строки, применяются фигурные скобки {}. То есть при выводе строки на консоль выражение {name} будет заменяться на значение переменной name - введенное имя. Однако чтобы можно было вводить таким образом значения переменных внутрь строки, перед строкой указывается знак доллара \$.



каталоге **bin/Debug** (оно будет называться по имени проекта и иметь расширение exe) и затем уже запускать без Visual Studio

Установка версии языка

Project -> Properties



Program.cs

FirstApp

Application

Build

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Configuration: Active (Debug)

Platform: Active (Any CPU)

Build Events:

Output file:

bin\Debug\

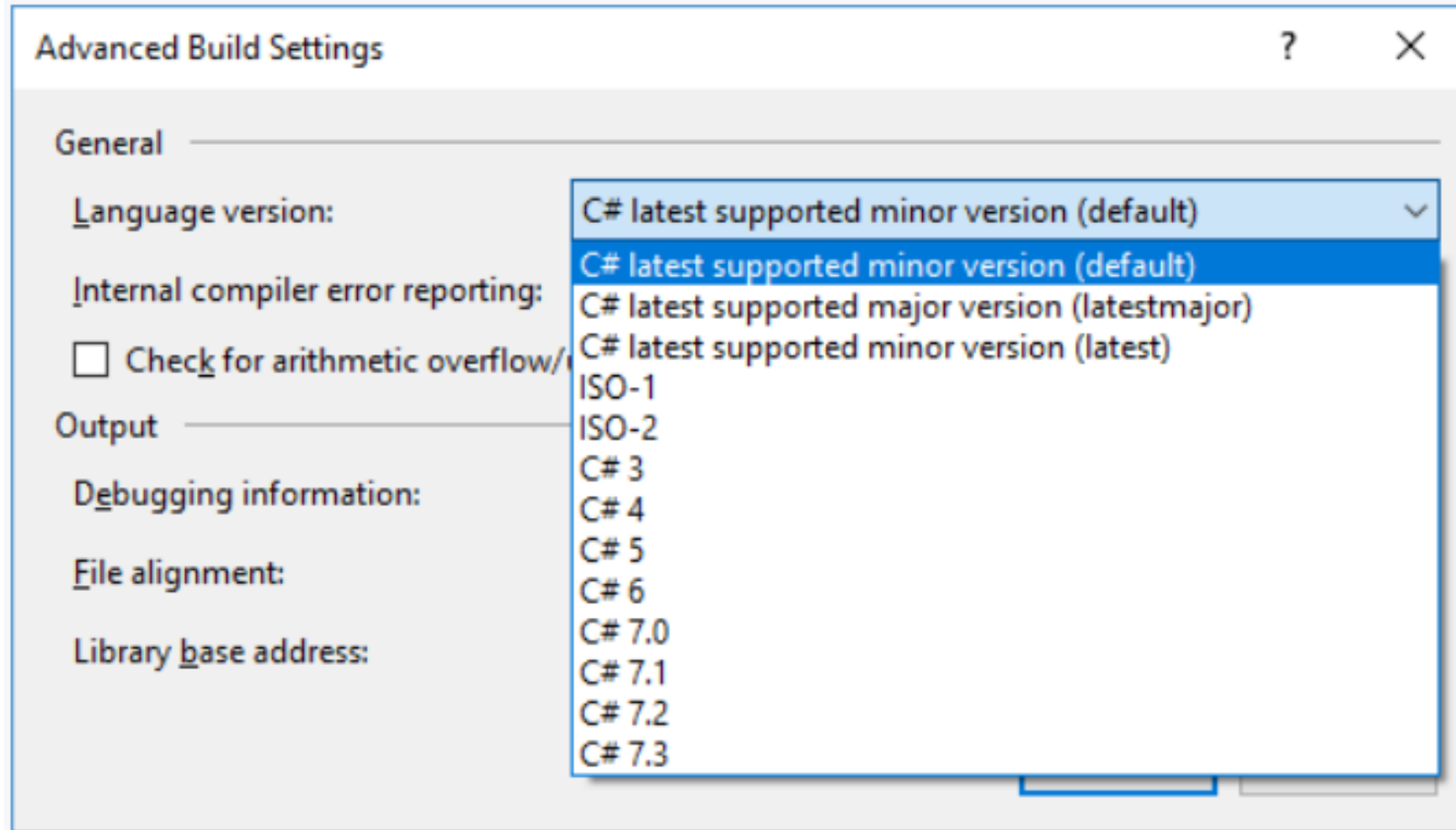
Browse...

COM interop

Output assembly:

Auto

Advanced...



если необходимо использовать самые последние нововведения языка, то следует установить опцию **C# latest minor version (latest)**

Структура программы

Инструкции

```
1 Console.WriteLine("Привет");
```

Набор инструкций может объединяться в блок кода.

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

Одни блоки кода могут содержать другие блоки:

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


Метод Main

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          // здесь помещаются выполняемые инструкции
6      }
7  }
```

Регистрозависимость

"Main"

"main"

"MAIN"

Комментарии

Однострочный комментарий размещается на одной строке после двойного слеша //. А многострочный комментарий заключается между символами /* текст комментария */.

```
1  using System;
2
3  namespace HelloApp
4  {
5      /*
6          программа, которая спрашивает у пользователя имя
7          и выводит его на консоль
8      */
9      class Program
10     {
11         // метод Main - стартовая точка приложения
12         static void Main(string[] args)
13         {
14             Console.Write("Введите свое имя: ");
15             string name = Console.ReadLine();        // вводим имя
16             Console.WriteLine($"Привет {name}");      // выводим имя на консоль
17             Console.ReadKey();
18         }
19     }
20 }
```

Переменные

```
1 тип имя_переменной;
```

- имя может содержать любые цифры, буквы и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания
- в имени не должно быть знаков пунктуации и пробелов
- имя не может быть ключевым словом языка C#

```
1 string name;  
2 string Name;
```

После определения переменной можно присвоить некоторое значение:

```
1 string name;  
2 name = "Tom";
```

```
1  using System;
2
3  namespace HelloApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string name = "Tom"; // определяем переменную и инициализируем ее
10
11              Console.WriteLine(name);    // Tom
12
13              name = "Bob";               // меняем значение переменной
14              Console.WriteLine(name);    // Bob
15
16              Console.Read();
17          }
18      }
19  }
```

Tom

Bob

Литералы

Логические литералы

Есть две логических константы - **true** (истина) и **false** (ложь):

```
1 Console.WriteLine(true);  
2 Console.WriteLine(false);
```

Целочисленные литералы

```
1 Console.WriteLine(-11);  
2 Console.WriteLine(5);  
3 Console.WriteLine(505);
```

```
1 Console.WriteLine(0b11);           // 3  
2 Console.WriteLine(0b1011);        // 11  
3 Console.WriteLine(0b100001);      // 33
```

```
1 Console.WriteLine(0x0A);           // 10  
2 Console.WriteLine(0xFF);           // 255  
3 Console.WriteLine(0xA1);           // 161
```


Вещественные литералы

```
1 3.14
2 100.001
3 -0.38
```

```
1 Console.WriteLine(3.2e3);
2 Console.WriteLine(1.2E-1);
```

Символьные литералы

```
1 '2'
2 'A'
3 'T'
```

управляющие последовательности

- '\n' - перевод строки
- '\t' - табуляция
- '\\' - обратный слеш

ASCII

```
1 Console.WriteLine('\x78'); // x
2 Console.WriteLine('\x5A'); // Z
```

Unicode

```
1 Console.WriteLine('\u0420'); // Р
2 Console.WriteLine('\u0421'); // С
```

Строковые литералы

```
1 Console.WriteLine("hello");  
2 Console.WriteLine("фыва");  
3 Console.WriteLine("hello word");
```

```
1 Console.WriteLine("Компания \"Рога и копыта\"");
```

```
1 Console.WriteLine("Привет \nмир");
```

null

null представляет ссылку, которая не указывает ни на какой объект.

Типы данных

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

```
1 bool alive = true;  
2 bool isDead = false;
```

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

```
1 byte bit1 = 1;  
2 byte bit2 = 102;
```

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

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

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

```
1 short n1 = 1;  
2 short n2 = 102;
```

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

```
1 ushort n1 = 1;  
2 ushort n2 = 102;
```

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

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

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

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

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

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

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

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

- **float**: хранит число с плавающей точкой от $-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$ и занимает 4 байта. Представлен системным типом `System.Single`
- **double**: хранит число с плавающей точкой от $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ и занимает 8 байта. Представлен системным типом `System.Double`
- **decimal**: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от $\pm 1.0 \cdot 10^{-28}$ до $\pm 7.9228 \cdot 10^{28}$, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом `System.Decimal`
- **char**: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом `System.Char`. Этому типу соответствуют символьные литералы:

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

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

```
1 string hello = "Hello";  
2 string word = "world";
```

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

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

```
1  using System;
2
3  namespace HelloApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string name = "Tom";
10             int age = 33;
11             bool isEmployed = false;
12             double weight = 78.65;
13
14             Console.WriteLine($"Имя: {name}");
15             Console.WriteLine($"Возраст: {age}");
16             Console.WriteLine($"Вес: {weight}");
17             Console.WriteLine($"Работает: {isEmployed}");
18         }
19     }
20 }
```

Имя: Tom

Возраст: 33

Вес: 78,65

Работает: False

Использование суффиксов

```
1 float a = 3.14F;  
2 float b = 30.6f;  
3  
4 decimal c = 1005.8M;  
5 decimal d = 334.8m;
```

```
1 uint a = 10U;  
2 long b = 20L;  
3 ulong c = 30UL;
```

Консольный ввод-вывод

Консольный вывод

```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             string hello = "Привет мир";
10            Console.WriteLine(hello);
11            Console.WriteLine("Добро пожаловать в C#!");
12            Console.WriteLine("Пока мир...");
13            Console.WriteLine(24.5);
14
15            Console.ReadKey();
16        }
17    }
18 }
```

Привет мир!

Добро пожаловать в C#!

Пока мир...

24,5

```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             string name = "Tom";
10            int age = 34;
11            double height = 1.7;
12            Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");
13
14            Console.ReadKey();
15        }
16    }
17 }
```

Имя: Tom Возраст: 34 Рост: 1,7м

```
1  using System;
2
3  namespace HelloApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string name = "Tom";
10             int age = 34;
11             double height = 1.7;
12             Console.WriteLine("Имя: {0}  Возраст: {2}  Рост: {1}м", name, height, age);
13
14             Console.ReadKey();
15         }
16     }
17 }
```

Консольный ввод

```
1  using System;
2
3  namespace HelloApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.Write("Введите свое имя: ");
10             string name = Console.ReadLine();
11             Console.WriteLine($"Привет {name}");
12
13             Console.ReadKey();
14         }
15     }
16 }
```

Введите свое имя: Том

Привет Том

- **Convert.ToInt32()** (преобразует к типу int)
- **Convert.ToDouble()** (преобразует к типу double)
- **Convert.ToDecimal()** (преобразует к типу decimal)

```
static void Main(string[] args)
{
    Console.Write("Введите имя: ");
    string name = Console.ReadLine();

    Console.Write("Введите возраст: ");
    int age = Convert.ToInt32(Console.ReadLine());

    Console.Write("Введите рост: ");
    double height = Convert.ToDouble(Console.ReadLine());

    Console.Write("Введите размер зарплаты: ");
    decimal salary = Convert.ToDecimal(Console.ReadLine());

    Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м  Зарплата: {salary}$");

    Console.ReadKey();
}
```

Введите имя: Том

Введите возраст: 25

Введите рост: 1,75

Введите размер зарплаты: 300,67

Имя: Том Возраст: 25 Рост: 1,75м

Зарплата: 300,67\$

Арифметические операции языка C#

- +

Операция сложения двух чисел:

```
1 int x = 10;  
2 int z = x + 12; // 22
```

- -

Операция вычитания двух чисел:

```
1 int x = 10;  
2 int z = x - 6; // 4
```

- *

Операция умножения двух чисел:

```
1 int x = 10;  
2 int z = x * 5; // 50
```

- /

операция деления двух чисел:

```
1 int x = 10;  
2 int z = x / 5; // 2  
3  
4 double a = 10;  
5 double b = 3;  
6 double c = a / b; // 3.33333333
```

- %

Операция получение остатка от целочисленного деления двух чисел:

```
1 double x = 10.0;  
2 double z = x % 4.0; //результат равен 2
```

- ++

Операция инкремента

```
1 int x1 = 5;  
2 int z1 = ++x1; // z1=6; x1=6  
3 Console.WriteLine($"{x1} - {z1}");  
4  
5 int x2 = 5;  
6 int z2 = x2++; // z2=5; x2=6  
7 Console.WriteLine($"{x2} - {z2}");
```

- --

```
1 int x1 = 5;  
2 int z1 = --x1; // z1=4; x1=4  
3 Console.WriteLine($"{x1} - {z1}");  
4  
5 int x2 = 5;  
6 int z2 = x2--; // z2=5; x2=4  
7 Console.WriteLine($"{x2} - {z2}");
```

Операции присваивания

- **+=**: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: выражение **A += B** равнозначно выражению **A = A + B**
- **-=**: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: **A -= B** эквивалентно **A = A - B**
- ***=**: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: **A *= B** эквивалентно **A = A * B**
- **/=**: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: **A /= B** эквивалентно **A = A / B**
- **%=**: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: **A %= B** эквивалентно **A = A % B**
- **&=**: присваивание после поразрядной конъюнкции. Присваивает левому операнду результат поразрядной конъюнкции его битового представления с битовым представлением правого операнда: **A &= B** эквивалентно **A = A & B**
- **|=**: присваивание после поразрядной дизъюнкции. Присваивает левому операнду результат поразрядной дизъюнкции его битового представления с битовым представлением правого операнда: **A |= B** эквивалентно **A = A | B**
- **^=**: присваивание после операции исключающего ИЛИ. Присваивает левому операнду результат операции исключающего ИЛИ его битового представления с битовым представлением правого операнда: **A ^= B** эквивалентно **A = A ^ B**
- **<<=**: присваивание после сдвига разрядов влево. Присваивает левому операнду результат сдвига его битового представления влево на определенное количество разрядов, равное значению правого операнда: **A <<= B** эквивалентно **A = A << B**
- **>>=**: присваивание после сдвига разрядов вправо. Присваивает левому операнду результат сдвига его битового представления вправо на определенное количество разрядов, равное значению правого операнда: **A >>= B** эквивалентно **A = A >> B**

Применение операций присвоения:

```
1 int a = 10;  
2 a += 10;      // 20  
3 a -= 4;       // 16  
4 a *= 2;       // 32  
5 a /= 8;       // 4  
6 a <<= 4;      // 64  
7 a >>= 2;      // 16
```

Операции присвоения являются правоассоциативными, то есть выполняются справа налево. Например:

```
1 int a = 8;  
2 int b = 6;  
3 int c = a += b -= 5;    // 9
```

В данном случае выполнение выражения будет идти следующим образом:

1. $b -= 5$ ($6-5=1$)
2. $a += (b-=5)$ ($8+1 = 9$)
3. $c = (a += (b-=5))$ ($c = 9$)

Преобразования базовых типов данных

Сужающие и расширяющие преобразования

```
1 byte a = 4;           // 0000100
2 ushort b = a;        // 000000000000100
```

В данном случае переменной типа `ushort` присваивается значение типа `byte`. Тип `byte` занимает 1 байт (8 бит), и значение переменной `a` в двоичном виде можно представить как:

```
1 00000100
```

Значение типа `ushort` занимает 2 байта (16 бит). И при присвоении переменной `b` значение переменной `a` расширяется до 2 байт

```
1 0000000000000100
```

То есть значение, которое занимает 8 бит, **расширяется** до 16 бит.

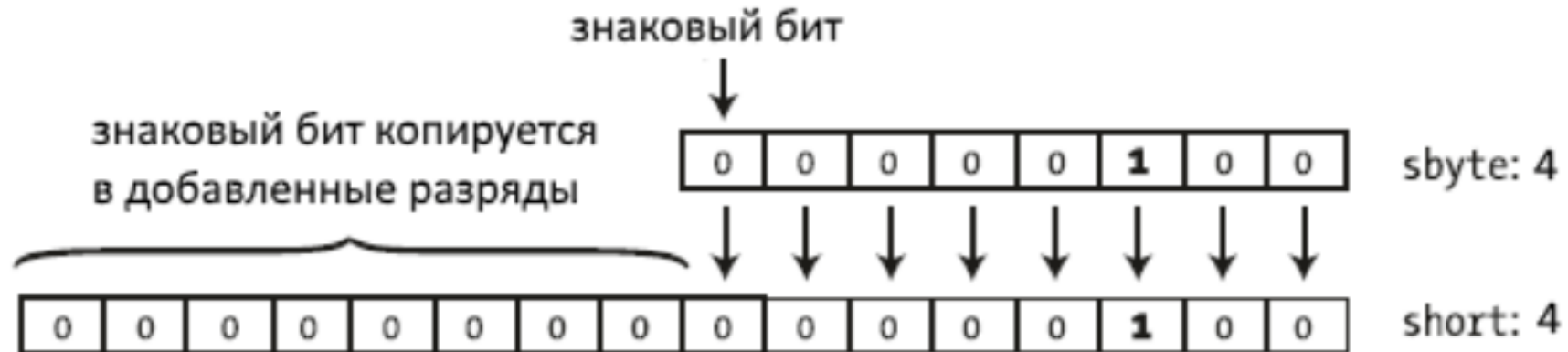
```
1 ushort a = 4;  
2 byte b = (byte) a;
```

Здесь переменной `b`, которая занимает 8 бит, присваивается значение `ushort`, которое занимает 16 бит. То есть из `00000000000000100` получаем `00000100`. Таким образом, значение сужается с 16 бит (2 байт) до 8 бит (1 байт).

Неявные преобразования

implicit conversion

```
1 sbyte a = 4;           // 0000100
2 short b = a;           // 000000000000100
```



```
1 int a = 4;  
2 int b = 6;  
3 byte c = (byte)(a+b);
```

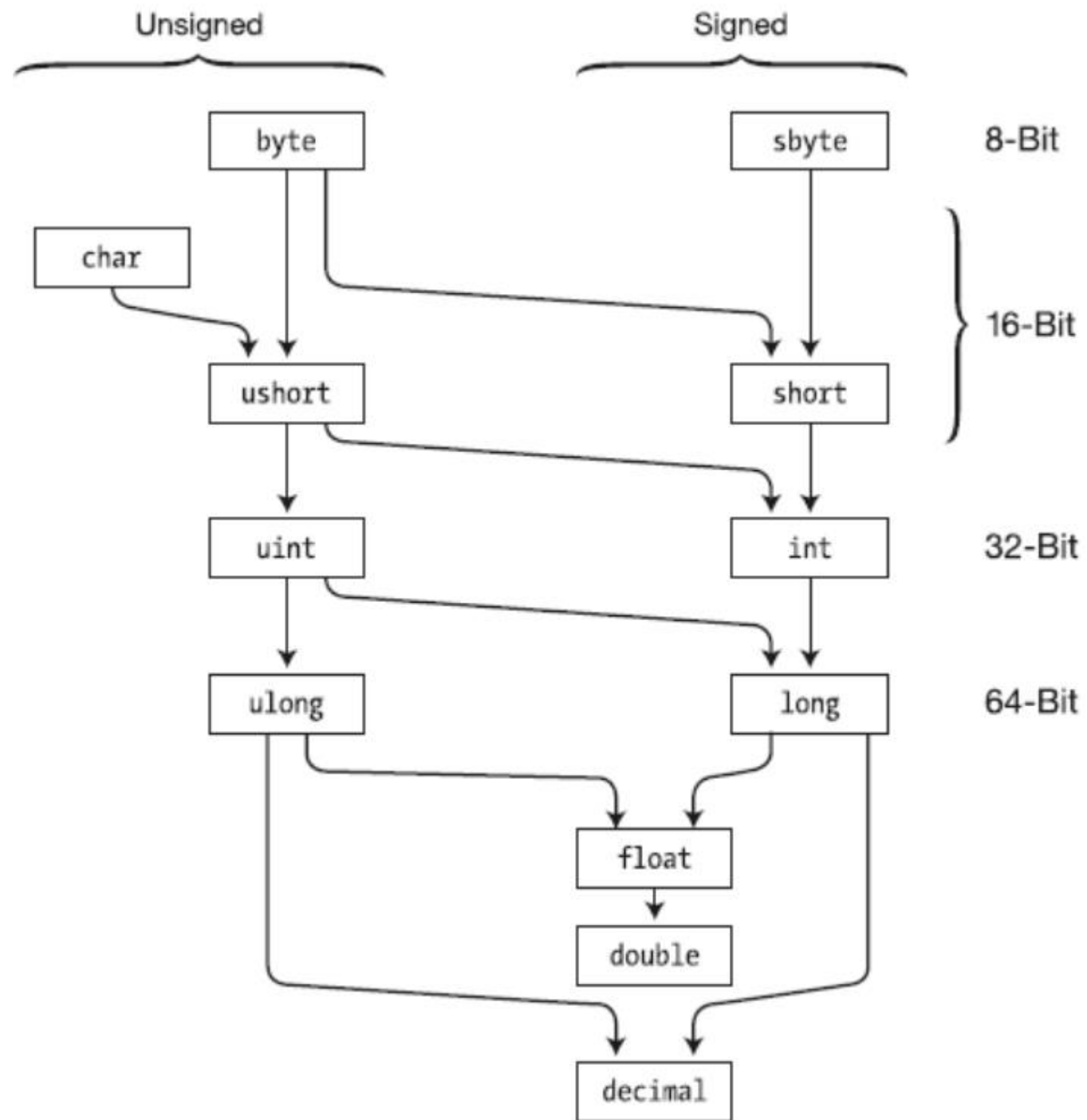
Расширяющие преобразования от типа с меньшей разрядностью к типу с большей разрядностью компилятор поводит неявно. Это могут быть следующие цепочки преобразований:

byte -> short -> int -> long -> decimal

int -> double

short -> float -> double

char -> int



Условные выражения

Операции сравнения

- **==**

Сравнивает два операнда на равенство. Если они равны, то операция возвращает **true**, если не равны, то возвращается **false**:

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a == b; // false
```

- **!=**

Сравнивает два операнда и возвращает true, если операнды не равны, и false, если они равны.

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a != b;    // true  
4 bool d = a!=10;     // false
```

- **<**

Операция "меньше чем". Возвращает true, если первый операнд меньше второго, и false, если первый операнд больше второго:

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a < b; // false
```

- >

Операция "больше чем". Сравнивает два операнда и возвращает true, если первый операнд больше второго, иначе возвращает false:

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a > b;    // true  
4 bool d = a > 25;   // false
```

- <=

Операция "меньше или равно". Сравнивает два операнда и возвращает true, если первый операнд меньше или равен второму. Иначе возвращает false.

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a <= b;    // false  
4 bool d = a <= 25;   // true
```

- >=

Операция "больше или равно". Сравнивает два операнда и возвращает true, если первый операнд больше или равен второму, иначе возвращается false:

```
1 int a = 10;  
2 int b = 4;  
3 bool c = a >= b;    // true  
4 bool d = a >= 25;   // false
```

Логические операции

- |

Операция логического сложения или логическое ИЛИ. Возвращает true, если хотя бы один из операндов возвращает true.

```
1 bool x1 = (5 > 6) | (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true
2 bool x2 = (5 > 6) | (4 > 6); // 5 > 6 - false, 4 > 6 - false, поэтому возвращается false
```

- &

Операция логического умножения или логическое И. Возвращает true, если оба операнда одновременно равны true.

```
1 bool x1 = (5 > 6) & (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается false
2 bool x2 = (5 < 6) & (4 < 6); // 5 < 6 - true, 4 < 6 - true, поэтому возвращается true
```

- ||

Операция логического сложения. Возвращает true, если хотя бы один из операндов возвращает true.

```
1 bool x1 = (5 > 6) || (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true
2 bool x2 = (5 > 6) || (4 > 6); // 5 > 6 - false, 4 > 6 - false, поэтому возвращается false
```

- **&&**

Операция логического умножения. Возвращает true, если оба операнда одновременно равны true.

```
1 bool x1 = (5 > 6) && (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается false
2 bool x2 = (5 < 6) && (4 < 6); // 5 < 6 - true, 4 < 6 - true, поэтому возвращается true
```

- **!**

Операция логического отрицания. Производится над одним операндом и возвращает true, если операнд равен false. Если операнд равен true, то операция возвращает false:

```
1 bool a = true;
2 bool b = !a;    // false
```

- **^**

Операция исключающего ИЛИ. Возвращает true, если либо первый, либо второй операнд (но не одновременно) равны true, иначе возвращает false

```
1 bool x5 = (5 > 6) ^ (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true
2 bool x6 = (50 > 6) ^ (4 / 2 < 3); // 50 > 6 - true, 4/2 < 3 - true, поэтому возвращается false
```


Условные конструкции

Конструкция if/else

```
1 int num1 = 8;  
2 int num2 = 6;  
3 if(num1 > num2)  
4 {  
5     Console.WriteLine($"Число {num1} больше числа {num2}");  
6 }
```

```
1 int num1 = 8;  
2 int num2 = 6;  
3 if(num1 > num2)  
4 {  
5     Console.WriteLine($"Число {num1} больше числа {num2}");  
6 }  
7 else  
8 {  
9     Console.WriteLine($"Число {num1} меньше числа {num2}");  
10 }
```

```
1  int num1 = 8;
2  int num2 = 6;
3  if(num1 > num2)
4  {
5      Console.WriteLine($"Число {num1} больше числа {num2}");
6  }
7  else if (num1 < num2)
8  {
9      Console.WriteLine($"Число {num1} меньше числа {num2}");
10 }
11 else
12 {
13     Console.WriteLine("Число num1 равно числу num2");
14 }
```

```
1  int num1 = 8;
2  int num2 = 6;
3  if(num1 > num2 && num1==8)
4  {
5      Console.WriteLine($"Число {num1} больше числа {num2}");
6  }
```

Конструкция switch

```
1 Console.WriteLine("Нажмите Y или N");
2 string selection = Console.ReadLine();
3 switch (selection)
4 {
5     case "Y":
6         Console.WriteLine("Вы нажали букву Y");
7         break;
8     case "N":
9         Console.WriteLine("Вы нажали букву N");
10        break;
11    default:
12        Console.WriteLine("Вы нажали неизвестную букву");
13        break;
14 }
```

```
1 int number = 1;
2 switch (number)
3 {
4     case 1:
5         Console.WriteLine("case 1");
6         goto case 5; // переход к case 5
7     case 3:
8         Console.WriteLine("case 3");
9         break;
10    case 5:
11        Console.WriteLine("case 5");
12        break;
13    default:
14        Console.WriteLine("default");
15        break;
16 }
```

Тернарная операция

[первый операнд – условие] ? [второй операнд] : [третий операнд]

```
1  int x=3;  
2  int y=2;  
3  Console.WriteLine("Нажмите + или -");  
4  string selection = Console.ReadLine();  
5  
6  int z = selection=="+"? (x+y) : (x-y);  
7  Console.WriteLine(z);
```

Циклы

- for
- foreach
- while
- do...while

Цикл for

```
1 for ([инициализация счетчика]; [условие]; [изменение счетчика])
2 {
3     // действия
4 }
```

```
1 for (int i = 0; i < 9; i++)
2 {
3     Console.WriteLine($"Квадрат числа {i} равен {i*i}");
4 }
```

```
1 int i = 0;
2 for (; ; )
3 {
4     Console.WriteLine($"Квадрат числа {++i} равен {i * i}");
5 }
```

```
1 int i = 0;
2 for (; i<9;)
3 {
4     Console.WriteLine($"Квадрат числа {++i} равен {i * i}");
5 }
```

Цикл do

```
1  int i = 6;  
2  do  
3  {  
4      Console.WriteLine(i);  
5      i--;  
6  }  
7  while (i > 0);
```

```
1  int i = -1;  
2  do  
3  {  
4      Console.WriteLine(i);  
5      i--;  
6  }  
7  while (i > 0);
```


Цикл while

```
1 int i = 6;
2 while (i > 0)
3 {
4     Console.WriteLine(i);
5     i--;
6 }
```

Операторы continue и break

```
1 for (int i = 0; i < 9; i++)
2 {
3     if (i == 5)
4         break;
5     Console.WriteLine(i);
6 }
```

0
1
2
3
4

```
1 for (int i = 0; i < 9; i++)
2 {
3     if (i == 5)
4         continue;
5     Console.WriteLine(i);
6 }
```

0
1
2
3
4
6
7
8

Массивы

```
1 тип_переменной[] название_массива;
```

```
1 int[] numbers;
```

```
1 int[] nums = new int[4];
```

```
1 int[] nums2 = new int[4] { 1, 2, 3, 5 };  
2  
3 int[] nums3 = new int[] { 1, 2, 3, 5 };  
4  
5 int[] nums4 = new[] { 1, 2, 3, 5 };  
6  
7 int[] nums5 = { 1, 2, 3, 5 };
```

```
1 int[] nums = new int[4];  
2 nums[0] = 1;  
3 nums[1] = 2;  
4 nums[2] = 3;  
5 nums[3] = 5;  
6 Console.WriteLine(nums[3]); // 5
```

Перебор массивов. Цикл foreach

```
1  foreach (тип_данных название_переменной in контейнер)
2  {
3      // действия
4  }
```

```
1  int[] numbers = new int[] { 1, 2, 3, 4, 5 };
2  foreach (int i in numbers)
3  {
4      Console.WriteLine(i);
5  }
```

```
1  int[] numbers = new int[] { 1, 2, 3, 4, 5 };
2  for (int i = 0; i < numbers.Length; i++)
3  {
4      Console.WriteLine(numbers[i]);
5  }
```

Многомерные массивы

```
1 int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };  
2  
3 int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Одномерный массив nums1

0	1	2	3	4	5
---	---	---	---	---	---

Двухмерный массив nums2

0	1	2
3	4	5

```
1 int[,] nums1;  
2 int[,] nums2 = new int[2, 3];  
3 int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };  
4 int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };  
5 int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };  
6 int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

```
1 int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };
2 foreach (int i in mas)
3     Console.Write($"{i} ");
4 Console.WriteLine();
```

```
1 int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };
2
3 int rows = mas.GetUpperBound(0) + 1;
4 int columns = mas.Length / rows;
5 // или так
6 // int columns = mas.GetUpperBound(1) + 1;
7
8 for (int i = 0; i < rows; i++)
9 {
10     for (int j = 0; j < columns; j++)
11     {
12         Console.Write($"{mas[i, j]} \t");
13     }
14     Console.WriteLine();
15 }
```

1	2	3
4	5	6
7	8	9
10	11	12

Массив массивов

```
1 int[][] nums = new int[3][];  
2 nums[0] = new int[2] { 1, 2 };           // выделяем память для первого подмассива  
3 nums[1] = new int[3] { 1, 2, 3 };        // выделяем память для второго подмассива  
4 nums[2] = new int[5] { 1, 2, 3, 4, 5 };  // выделяем память для третьего подмассива
```

Зубчатый массив nums

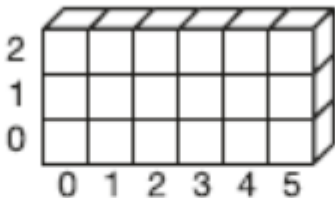
1	2			
1	2	3		
1	2	3	4	5

Одномерный массив

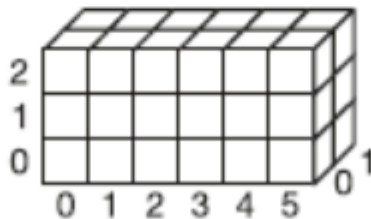


Одномерный массив
int[5]

Многомерные массивы

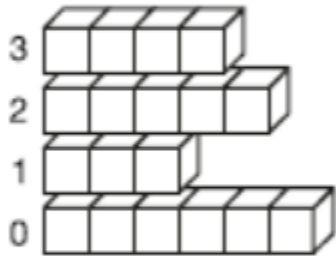


Двухмерный массив
int[3,6]



Трёхмерный массив
int[3,6,2]

Зубчатый массив



Зубчатый массив
int[4][]

Основные понятия массивов

Суммирую основные понятия массивов:

- **Ранг** (rank): количество измерений массива
- **Длина измерения** (dimension length): длина отдельного измерения массива
- **Длина массива** (array length): количество всех элементов массива

Например, возьмем массив

```
1 | int[,] numbers = new int[3, 4];
```

Массив numbers двумерный, то есть он имеет два измерения, поэтому его ранг равен 2. Длина первого измерения - 3, длина второго измерения - 4. Длина массива (то есть общее количество элементов) - 12.

Методы

```
1 [модификаторы] тип_возвращаемого_значения название_метода ([параметры])
2 {
3     // тело метода
4 }
```

```
1 static void Main(string[] args)
2 {
3
4 }
```

```
namespace HelloApp
{
    class Program
    {
        static void Main(string[] args)
        {

        }

        static void SayHello()
        {
            Console.WriteLine("Hello");
        }
        static void SayGoodbye()
        {
            Console.WriteLine("GoodBye");
        }
    }
}
```

Вызов методов

1 | название_метода (значения_для_параметров_метода);

```
class Program
{
    static void Main(string[] args)
    {
        SayHello();
        SayGoodbye();

        Console.ReadKey();
    }

    static void SayHello()
    {
        Console.WriteLine("Hello");
    }
    static void SayGoodbye()
    {
        Console.WriteLine("GoodBye");
    }
}
```

Hello

GoodBye

Возвращение значения

```
1 return возвращаемое значение;
```

```
1 static string GetHello()  
2 {  
3     return "Hello";  
4 }  
5 static int GetSum()  
6 {  
7     int x = 2;  
8     int y = 3;  
9     int z = x + y;  
10    return z;  
11 }
```

```
1 static int GetSum()  
2 {  
3     int x = 2;  
4     int y = 3;  
5     return x + y;  
6 }
```

```
1 static int GetSum()  
2 {  
3     int x = 2;  
4     int y = 3;  
5     return "5"; // ошибка - надо возвращать число  
6 }
```

```
1  using System;
2
3  namespace HelloApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string message = GetHello();
10             int sum = GetSum();
11
12             Console.WriteLine(message); // Hello
13             Console.WriteLine(sum);    // 5
14
15             Console.ReadKey();
16         }
17
```

```
18     static string GetHello()
19     {
20         return "Hello";
21     }
22     static int GetSum()
23     {
24         int x = 2;
25         int y = 3;
26         return x + y;
27     }
28 }
29
```

```
1 static string GetHello()  
2 {  
3     return "Hello";  
4     Console.WriteLine("After return");  
5 }
```

```
1 static void SayHello()  
2 {  
3     int hour = 23;  
4     if(hour > 22)  
5     {  
6         return;  
7     }  
8     else  
9     {  
10        Console.WriteLine("Hello");  
11    }  
12 }
```

Сокращенная запись методов

```
1 static void SayHello()  
2 {  
3     Console.WriteLine("Hello");  
4 }
```

```
1 static void SayHello() => Console.WriteLine("Hello");
```

```
1 static string GetHello()  
2 {  
3     return "hello";  
4 }
```

```
1 static string GetHello() => "hello";
```

Параметры методов

```
1 static int Sum(int x, int y)
2 {
3     return x + y;
4 }
```

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int result = Sum(10, 15);
6         Console.WriteLine(result); // 25
7
8         Console.ReadKey();
9     }
10    static int Sum(int x, int y)
11    {
12        return x + y;
13    }
14 }
```



```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         Display("Tom", 24); // Name: Tom Age: 24
6
7         Console.ReadKey();
8     }
9     static void Display(string name, int age)
10    {
11        Console.WriteLine($"Name: {name} Age: {age}");
12    }
13 }
```

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int a;
6         int b = 9;
7         Sum(a, b); // Ошибка - переменной a не присвоено значение
8
9         Console.ReadKey();
10    }
11    static int Sum(int x, int y)
12    {
13        return x + y;
14    }
15 }
```

Необязательные параметры

```
1 static int OptionalParam(int x, int y, int z=5, int s=4)
2 {
3     return x + y + z + s;
4 }
```

Именованные параметры

```
1 static int OptionalParam(int x, int y, int z=5, int s=4)
2 {
3     return x + y + z + s;
4 }
5 static void Main(string[] args)
6 {
7     OptionalParam(x:2, y:3);
8
9     //Необязательный параметр z использует значение по умолчанию
10    OptionalParam(y:2, x:3, s:10);
11
12    Console.ReadKey();
13 }
```

```
1 static void Main(string[] args)
2 {
3     OptionalParam(2, 3);
4
5     OptionalParam(2,3,10);
6
7     Console.ReadKey();
8 }
```

Передача параметров по ссылке и значению. Выходные параметры

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Sum(10, 15);          // параметры передаются по значению
6          Console.ReadKey();
7      }
8      static int Sum(int x, int y)
9      {
10         return x + y;
11     }
12 }
```

Передача параметров по ссылке и модификатор ref

```
1  static void Main(string[] args)
2  {
3      int x = 10;
4      int y = 15;
5      Addition(ref x, y); // ВЫЗОВ МЕТОДА
6      Console.WriteLine(x); // 25
7
8      Console.ReadLine();
9  }
10 // параметр x передается по ссылке
11 static void Addition(ref int x, int y)
12 {
13     x += y;
14 }
```

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int a = 5;
6         Console.WriteLine($"Начальное значение переменной a = {a}");
7
8         //Передача переменных по значению
9         //После выполнения этого кода по-прежнему a = 5, так как мы передали лишь ее копию
10        IncrementVal(a);
11        Console.WriteLine($"Переменная a после передачи по значению равна = {a}");
12        Console.ReadKey();
13    }
14    // передача по значению
15    static void IncrementVal(int x)
16    {
17        x++;
18        Console.WriteLine($"IncrementVal: {x}");
19    }
20 }
```

Начальное значение переменной a = 5

IncrementVal: 6

Переменная a после передачи по значению
равна = 5

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int a = 5;
6         Console.WriteLine($"Начальное значение переменной a = {a}");
7         //Передача переменных по ссылке
8         //После выполнения этого кода a = 6, так как мы передали саму переменную
9         IncrementRef(ref a);
10        Console.WriteLine($"Переменная a после передачи ссылке равна = {a}");
11
12        Console.ReadKey();
13    }
14    // передача по ссылке
15    static void IncrementRef(ref int x)
16    {
17        x++;
18        Console.WriteLine($"IncrementRef: {x}");
19    }
20 }
```

Начальное значение переменной a = 5
IncrementRef: 6
Переменная a после передачи по ссылке
равна = 6

Выходные параметры. Модификатор out

```
1 static void Sum(int x, int y, out int a)
2 {
3     a = x + y;
4 }
```

```
1 static void Main(string[] args)
2 {
3     int x = 10;
4
5     int z;
6
7     Sum(x, 15, out z);
8
9     Console.WriteLine(z);
10
11     Console.ReadKey();
12 }
```

```
1 static void Main(string[] args)
2 {
3     int x = 10;
4     int area;
5     int perimetr;
6     GetData(x, 15, out area, out perimetr);
7     Console.WriteLine("Площадь : " + area);
8     Console.WriteLine("Периметр : " + perimetr);
9
10    Console.ReadKey();
11 }
12 static void GetData(int x, int y, out int area, out int perim)
13 {
14     area= x * y;
15     perim= (x + y)*2;
16 }
```


Входные параметры. Модификатор in

Модификатор in указывает, что через данный параметр будет передаваться в метод по ссылке, однако внутри метода его значение параметра нельзя будет изменить.

```
1 static void GetData(in int x, int y, out int area, out int perim)
2 {
3     // x = x + 10; нельзя изменить значение параметра x
4     y = y + 10;
5     area = x * y;
6     perim = (x + y) * 2;
7 }
```

Массив параметров и ключевое слово params

используя ключевое
слово **params**, мы можем
передавать неопределенное
количество параметров

```
1 static void Addition(params int[] integers)
2 {
3     int result = 0;
4     for (int i = 0; i < integers.Length; i++)
5     {
6         result += integers[i];
7     }
8     Console.WriteLine(result);
9 }
10
11 static void Main(string[] args)
12 {
13     Addition(1, 2, 3, 4, 5);
14
15     int[] array = new int[] { 1, 2, 3, 4 };
16     Addition(array);
17
18     Addition();
19     Console.ReadLine();
20 }
```

Если же нам надо передать какие-то другие параметры, то они должны указываться до параметра с ключевым словом `params`:

```
1 //Так работает
2 static void Addition( int x, string mes, params int[] integers)
3 {}
```

Вызов подобного метода:

```
1 Addition(2, "hello", 1, 3, 4);
```

Массив в качестве параметра

```
1  // передача параметра с params
2  static void Addition(params int[] integers)
3  {
4      int result = 0;
5      for (int i = 0; i < integers.Length; i++)
6      {
7          result += integers[i];
8      }
9      Console.WriteLine(result);
10 }
11 // передача массива
12 static void AdditionMas(int[] integers, int k)
13 {
14     int result = 0;
15     for (int i = 0; i < integers.Length; i++)
16     {
17         result += (integers[i]*k);
18     }
19     Console.WriteLine(result);
20 }
```

```
21
22 static void Main(string[] args)
23 {
24     Addition(1, 2, 3, 4, 5);
25
26     int[] array = new int[] { 1, 2, 3, 4 };
27     AdditionMas(array, 2);
28
29     Console.ReadLine();
30 }
```

Область видимости (контекст) переменных

- Контекст класса. Переменные, определенные на уровне класса, доступны в любом методе этого класса
- Контекст метода. Переменные, определенные на уровне метода, являются локальными и доступны только в рамках данного метода. В других методах они недоступны
- Контекст блока кода. Переменные, определенные на уровне блока кода, также являются локальными и доступны только в рамках данного блока. Вне своего блока кода они не доступны.

```
1  class Program
2  {
3      static int a = 9; // переменная уровня класса
4
5      static void Main(string[] args)
6      {
7          int a = 5; // скрывает переменную a, которая объявлена на уровне класса
8          Console.WriteLine(a); // 5
9      }
10 }
```

```
1 class Program // начало контекста класса
2 {
3     static int a = 9; // переменная уровня класса
4
5     static void Main(string[] args) // начало контекста метода Main
6     {
7         int b = a - 1; // переменная уровня метода
8
9         { // начало контекста блока кода
10
11             int c = b - 1; // переменная уровня блока кода
12
13         } // конец контекста блока кода, переменная c уничтожается
14
15         //так нельзя, переменная c определена в блоке кода
16         //Console.WriteLine(c);
17
18         //так нельзя, переменная d определена в другом методе
19         //Console.WriteLine(d);
20
```

```
21         Console.Read();
22
23     } // конец контекста метода Main, переменная b уничтожается
24
25     void Display() // начало контекста метода Display
26     {
27         // переменная a определена в контексте класса, поэтому доступна
28         int d = a + 1;
29
30     } // конец контекста метода Display, переменная d уничтожается
31
32 } // конец контекста класса, переменная a уничтожается
```