



Синтаксис С#

(сборки .net core, переменные,
основные типы данных)

Андрей Голяков

Полезные расширения для Visual Studio

Существует огромное количество расширений для IDE Visual Studio.

Они доступны по URL <https://marketplace.visualstudio.com> или через меню Tools > Extensions and Updates:

- GitHub Extension for Visual Studio
- File Icons
- GhostDoc
- BetterStartPage



Развертывание приложений .NET Core

По-умолчанию из Visual Studio приложение собирается для развертывания, **зависящего от платформы** (в англоязычной документации еще говорят **Portable** или **FDD**: Framework-Dependent Deployment).

- При таком виде сборки, в папке назначения мы получаем только файлы нашего приложения и внешних зависимостей (сторонних библиотек).
- Для запуска нашего приложения на целевом компьютере должен быть установлен **.NET Core Runtime** соответствующей версии.
** Runtime – для запуска приложений, не путать с SDK – для разработки!*
- Проверить, какие компоненты установлены в текущей системе можно с помощью команд:
 - **dotnet --list-runtimes**
 - **dotnet --list-sdks**



Развертывание приложений .NET Core

Также бывает автономное развертывание (в англоязычной документации его называют **Standalone** или **SCD**: Self-Contained Deployment).

При такой сборке все компоненты для выполнения и библиотеки .NET Core, и сторонние библиотеки, то есть **абсолютно все зависимости**, поставляются вместе с самим приложением (чаще всего в одной папке).

Можно пойти в папку проекта (где находится файл .csproj) и выполнить одну из двух команд ниже и заглянуть в папку `bin\Debug\netcoreappX.X\win10-x64\publish`

```
dotnet publish -r win10-x64
```

```
dotnet publish -r win10-x64 -p:PublishSingleFile=true
```



Запуск приложений .NET Core **в консоли**

Запустим наше приложение в консоли как отдельный файл сборки, а не через окружение Visual Studio.

Поскольку у нас **portable-сборка**, наше приложение выглядит как DLL-библиотека, хотя, и является исполняемым кодом с точкой входа.

Относительно солюшна сборка располагается по следующему пути:

папка_проекта\bin\Debug\netcoreappX.X

Запустить его можно с помощью команды dotnet :

dotnet имя_сборки.dll

Можно перед этим очистить экран командой **cls**.

** С версии .NET Core 3.0 также генерируется исполняемый файл для платформы, на которой происходит сборка (т.е. для Windows - привычный exe-файл)*



Синтаксис C#

- Заявления (**statements**) могут состоять из одного или нескольких выражений (**expressions**) или переменных (**variables**).
- Блоки (**blocks**) - несколько выражений или блоков, объединенных фигурными скобками.
- Комментарии (**comments**)

a. `//` однострочные: для выделенного текста можно использовать:
`// Ctrl + K + C` (закомментировать)
`// Ctrl + K + U` (раскомментировать)

b. `/*`
 многострочные
 комментарии
`*/`



Синтаксис C#

```
using System; // a semicolon indicates the end of a statement class
public class Program
{
    // start of external block
    static void Main()
    {
        // start of inner block
        Console.WriteLine("Hello World!"); // this is one statement
    }
    // end of inner block
}
// end of external block
```



Переменные

- Все приложения обрабатывают **данные**.
- Данные **приходят** откуда-то, **обрабатываются** и **уходят** куда-то.
- В процессе обработки данные могут храниться в **переменных** — именованных местах в памяти выполняемой программы.
- Память приложения - **временная**, т.е. при завершении приложения всё, что в ней хранилось, удаляется.



Буквальные значения

Конкретные неизменные значения любого типа данных, например:

- 12 целочисленное значение
- 9.99 дробное числовое значение
- 'D' символьное значение
- "Я – Андрей" строковое значение
- true значение булевого типа данных



Определение переменных

```
[тип данных] [имя переменной];  
[имя переменной] = [значение];
```

или

```
[тип данных] [имя переменной] = [значение];
```

Пример раздельного определения переменной и задания значения:

```
int a;  
a = 12;
```

Пример одновременного определения переменной с заданным значением:

```
int b = 13;
```



Правила именования переменных

Disclaimer :)

Эти правила не являются официальными, код будет компилироваться, а приложение работать и без учета правил оформления кода. Более того, в разных компаниях эти правила различны, однако следование одним правилам поможет быстрее адаптироваться к другим, а это, в свою очередь, повышает скорость чтения чужого кода в рамках одной экосистемы.

- Используйте **camelCase** для имен переменных
- При именовании переменных **избегайте использования сокращений** вроде `l`, `t` или `num`. Используйте `index`, `temp` или `number`
- Не используйте венгерскую нотацию!

Хорошая статья о стайл гайдах: [C#: требования и рекомендации по написанию кода](#)
(комментарии к статье не менее ценные, чем сама статья)



СИМВОЛЫ И СТРОКИ

СИМВОЛЫ

System.Char или ключевое слово `char`

```
char letter = 'A'; // declaring a single-char variable  
Console.WriteLine(letter);
```

Строки

System.String или ключевое слово `string`

```
string name = "Bob"; // declaring a string variable  
Console.WriteLine(name);
```



Целые числа

Однobaйтовое целое число

System.Byte или ключевое слово `byte` (0 – 255)

```
byte age = 36;  
Console.WriteLine(age);  
byte ageInHex = 0x24;  
Console.WriteLine(ageInHex);
```

System.SByte или ключевое слово `sbyte` (-128 – 127)

```
sbyte min = -128;  
Console.WriteLine(min);
```

Двухбайтовое целое число

System.Int16 или ключевое слово `short` (-32 768 – 32 767)

```
short pressure = -21200;  
Console.WriteLine(pressure);
```

System.UInt16 или ключевое слово `ushort` (0 – 65 535)

```
ushort yearOfBirth = 1982;  
Console.WriteLine(yearOfBirth);
```



Целые числа

Четырехбайтовое целое число

System.Int32 или ключевое слово `int` (-2 147 483 648 – 2 147 483 647)

```
int minutesInYear = 365 * 24 * 60;  
Console.WriteLine(minutesInYear);
```

System.UInt32 или ключевое слово `uint` (0 – 4 294 967 295)

```
int minutesInYear = 365 * 24 * 60;  
Console.WriteLine(minutesInYear);
```

Восьмибайтовое целое число

System.Int64 или ключевое слово `long` (-9 223 372 036 854 775 808 – 9 223 372 036 854 775 807)

```
long yearOfBirth = 1982;  
Console.WriteLine(yearOfBirth);
```

System.UInt64 или ключевое слово `ulong` (0 to 18 446 744 073 709 551 615)

```
ulong nextToMaxLong = 9223372036854775807;  
Console.WriteLine(nextToMaxLong);
```



Числа с плавающей точкой

Четырехбайтовое дробное число (точность ~6–9 знаков после запятой)

`System.Single` или ключевое слово `float` ($\pm 1.5 \times 10^{-45}$ – $\pm 3.4 \times 10^{38}$)

```
float x = 3.5F;  
Console.WriteLine(x);
```

```
int x = 3;  
float y = 4.5f;  
short z = 5;  
var result = x * y / z;  
Console.WriteLine("The result is {0}", result);  
Type type = result.GetType();  
Console.WriteLine("result is of type {0}", type.ToString());
```



Числа с плавающей точкой

Восьмибайтовое дробное число (точность ~15–17 знаков после запятой)

`System.Double` или ключевое слово `double` ($\pm 5.0 \times 10^{-324}$ – $\pm 1.7 \times 10^{308}$)

```
double y = 3D;  
Console.WriteLine(y);  
  
// Mixing types in expressions  
int x = 3;  
float y = 4.5f;  
short z = 5;  
double w = 1.7E+3;  
// Result of the 2nd argument is a double:  
Console.WriteLine("The sum is {0}", x + y + z + w);  
  
// Output: The sum is 1712.5
```



Числа с плавающей точкой

16-байтное дробное число повышенной точности (~28–29 знаков после запятой)

`System.Decimal` или ключевое слово `decimal` ($\pm 1.0 \times 10^{-28}$ – $\pm 7.9228 \times 10^{28}$)

```
decimal myMoney = 300.5M;  
Console.WriteLine(myMoney);
```

```
decimal dec = 0m;  
double dub = 9;  
// The following line causes an error that reads "Operator '+' cannot  
// be applied to operands of type 'double' and 'decimal'"  
Console.WriteLine(dec + dub); // gives compile error!
```

```
// You can fix the error by using explicit casting of either operand.  
Console.WriteLine(dec + (decimal)dub);  
Console.WriteLine((double)dec + dub);
```



Распознавание числовых значений

Метод `Parse(...)` работает для всех числовых типов данных.

```
string s = "175";  
int i = int.Parse(s);  
Console.WriteLine(i + 25);
```

```
byte.Parse(...);  
sbyte.Parse(...);  
short.Parse(...);  
ushort.Parse(...);
```

```
int.Parse(...);  
uint.Parse(...);  
long.Parse(...);  
ulong.Parse(...);
```

```
float.Parse(...);  
double.Parse(...);  
decimal.Parse(...);
```



Булевы величины

`true` / `false` (или 1 / 0 или “Да” / “Нет” и т.д.)

`System.Boolean` или ключевое слово `bool` (`true` или `false`)

```
bool b = true;  
Console.WriteLine(b);
```

```
// Boolean operation AND  
Console.WriteLine(true && true);    // logical AND: T && T = T  
Console.WriteLine(true && false);   // logical AND: T && F = F  
Console.WriteLine(false && false);  // logical AND: F && F = F
```

```
// Boolean operation OR  
Console.WriteLine(true || true);    // logical OR: T || T = T  
Console.WriteLine(true || false);   // logical OR: T || F = T  
Console.WriteLine(false || false);  // logical OR: F || F = F
```



Операторы булевых величин

Операторы `==` , `!=` , `>` , `<` , `!`

```
// Result of comparison is a boolean type value
```

```
bool a = 12 > 17;
```

```
Console.WriteLine(a);
```

```
bool b = "one" == "two"; // Strings are different, result is False
```

```
Console.WriteLine(b);
```

```
bool c = "one" != "two"; // Strings are different, result is True
```

```
Console.WriteLine(c);
```

```
bool d = 12 != 12; // Numbers are the same, result is False
```

```
Console.WriteLine(!d); // We asked for NOT(d), result is True
```



Домашнее задание

Написать приложение, запрашивающее у пользователя поочерёдно 2 числа, а затем выводящее сумму, разницу и произведение этих чисел в консоль.

** Начиная с этого урока присылайте, пожалуйста, не просто ссылку на репозиторий, а ссылку на последний коммит с кодом домашнего задания. Это позволит в случае дополнительных вопросов (с моей стороны) и доделок (с вашей) удобно смотреть разницу между коммитами – что именно изменилось?*



Домашнее задание (со звёздочкой)

Вариант “посложнее”, если вы знакомы с условной конструкцией `if...else`:

Написать приложение-калькулятор, запрашивающее у пользователя поочерёдно 2 числа числа, а также один из шести типов операций:

- сложение
- вычитание
- умножение
- деление
- остаток от деления
- возведение в степень

а затем выводящее результат вычисления в консоль.

** Начиная с этого урока присылайте, пожалуйста, не просто ссылку на репозиторий, а ссылку на последний коммит с кодом домашнего задания. Это позволит, в случае дополнительных вопросов (с моей стороны) и доделок (с вашей), удобно смотреть разницу между коммитами – что именно изменилось?*



Спасибо за внимание.

