

# **Тема 3. Типы данных. Переменные. Константы. Комментарии. Литералы.**

# **Учебные вопросы:**

- 1. Установка Visual Studio.**
- 2. Встроенные типы данных C#.**
- 3. Переменные. Константы.**
- 4. Комментарии.**
- 5. Литералы.**

# 1. Установка Visual Studio.

**Visual Studio** - это полнофункциональная интегрированная среда разработки (IDE) от Microsoft, предоставляющая разработчикам широкий набор инструментов для создания высококачественных приложений. Она широко используется для разработки на C#, но также поддерживает другие языки программирования, такие как C++, VB.NET, Python, JavaScript и многие другие.

**IDE** — это комплексное программное решение, предоставляющее программистам единую среду для написания, тестирования и отладки кода. Она объединяет в себе такие инструменты, как продвинутый текстовый редактор с подсветкой синтаксиса, компилятор для преобразования кода в машинный язык и отладчик для обнаружения и исправления ошибок.

## Основные возможности Visual Studio:

### •Редактор кода:

- Подсветка синтаксиса, автодополнение кода, рефакторинг, навигация по коду.
- Интеллектуальное завершение кода, позволяющее быстро и точно вводить код.

### •Отладка:

- Пошаговое выполнение кода, установка точек прерывания, инспекция переменных.
- Возможность отладки как управляемого, так и неуправляемого кода.

### •Профилирование:

- Анализ производительности приложения для выявления узких мест.

### •Дизайн пользовательского интерфейса:

- Визуальные дизайнеры для создания форм Windows Forms, WPF, веб-приложений и мобильных приложений.

### •Управление проектами:

- Создание, сборка и развертывание проектов.
- Поддержка различных платформ и устройств.

### •Система контроля версий:

- Интеграция с популярными системами контроля версий, такими как Git.

### •Расширения:

- Возможность расширения функциональности с помощью тысяч доступных расширений.

# Установка Visual Studio

## Шаг 1: Скачать установщик

- **Перейдите на официальный сайт Microsoft:**  
<https://visualstudio.microsoft.com/ru/downloads/>
- **Выберите нужную версию Visual Studio (community)**
- **Нажмите кнопку "Скачать" и сохраните установщик на свой компьютер.**



visualstudio.microsoft.com

Скачать Инструменты Visual Studio — установит...



пересказать



## Visual Studio 2022



Наиболее полная интегрированная среда разработки для разработчиков .NET и C++ в Windows для создания веб-приложений, облачных, классических и мобильных приложений, а также служб и игр.

## версия

Получите ранний доступ к последним функциям, которые еще не доступны в основном выпуске

**Подробнее →**

### Community

Мощная интегрированная среда разработки, бесплатная для студентов, участников проектов с открытым кодом и отдельных пользователей

**Скачать бесплатно**

### Профессиональный

Профессиональная интегрированная среда разработки, оптимально подходящая для небольших команд

**Бесплатная пробная версия**

### Enterprise

Комплексное масштабируемое решение для команд любого размера

**Бесплатная пробная версия**

## Шаг 2: Запуск установщика

- **Запустите скачанный файл установщика** (VisualStudioSetup.exe).
- **Выберите рабочие нагрузки и компоненты:** Установщик предложит вам выбрать компоненты, которые вы хотите установить. Отметьте те, которые необходимы для ваших проектов (например, разработка для .NET, мобильная разработка, веб-разработка).
- **Настройте дополнительные параметры:** Вы можете настроить язык интерфейса, расположение установки и другие параметры.
- **Нажмите кнопку "Установить".**

## Рабочие нагрузки

Отдельные компоненты

Языковые пакеты

Расположения установки



### Разработка на Python

Редактирование, отладка, интерактивная разработка и система управления версиями для Python.



### Разработка Node.js

Создавайте масштабируемые сетевые приложения с помощью Node.js, асинхронной среды выполнения Ja...



## Классические и мобильные приложения (5)



### Разработка с помощью .NET Multi-Platform App UI

Создавайте приложения Android, iOS, Windows и Mac из общей базы кода на языке C# с помощью .NET MA...



### Разработка классических приложений .NET

Создание приложений WPF, Windows Forms и консольных приложений с использованием C#, Visual...



### Разработка классических приложений на C++

Создавайте современные приложения C++ для Windows с использованием любых удобных инструме...



### Разработка Windows-приложений

Выполняйте сборку приложений для платформы Windows, используя WinUI с C# или C++ (при необход...



### Разработка мобильных приложений на языке C++





## **Шаг 3: Ожидание установки**

Процесс установки может занять некоторое время в зависимости от выбранных компонентов и характеристик вашего компьютера.

## **Шаг 4: Запуск Visual Studio**

После завершения установки запустите Visual Studio.

## Шаг 5: Создание проекта в Visual Studio

При первом запуске Visual Studio откроет начальное окно. Здесь вы увидите различные шаблоны проектов.

### Выберите шаблон:

Visual Studio предлагает широкий выбор шаблонов для различных типов проектов.

Выберите консольное приложение (.NET framework) или консольное приложение Microsoft.

**.NET Framework:** Монолитная платформа, тесно интегрированная с Windows.

**Консольное приложение Microsoft** - модульная, кроссплатформенная платформа, работающая на Windows, Linux и macOS.

### Настройте проект:

Имя проекта: Придумайте понятное и уникальное имя.


Расположение: можно оставить как есть.


Framework: можно оставить как есть.

Нажмите "Создать".

# Создание проекта

## Последние шаблоны проектов

 Консольное приложение (Майкрософт) C#

 Консольное приложение (.NET Framework) C#

Поиск шаблонов (ALT+ "B")



Очистить все

C#

Все платформы

Все типы проектов



Консольное приложение (.NET Framework)  
Проект приложения для командной строки

C#

Windows

Консоль



Библиотека классов (.NET Framework)  
Проект для создания библиотеки классов C# (.dll)

C#

Windows

Библиотека



Веб-задание Azure (.NET Framework)  
Шаблон проекта для создания веб-заданий, с помощью которых можно запускать программы в своих веб-приложениях Azure.

C#

Azure

Облако



Проект модульного теста (.NET Framework)  
Проект, содержащий модульные тесты MSTest.

C#

Windows

Тестирование



Тестовый проект xUnit  
Проект с тестами xUnit.net, которые могут выполняться на базе .NET в Windows, Linux и MacOS.

C#

Linux

macOS

Windows

Тестирование



Тестирование веб-драйвера для Microsoft Edge (.NET Core)  
Проект, содержащий модульные тесты, которые позволяют автоматизировать

Назад

Далее

# Настроить новый проект

Консольное приложение (.NET Framework)

C#

Windows

Консоль

Имя проекта

MyFirstApp

Расположение

C:\Users\user\source\repos

Имя решения ⓘ

MyFirstApp

☐ Поместить решение и проект в одном каталоге

Платформа

.NET Framework 4.8

Проект будет создан в "C:\Users\user\source\repos\MyFirstApp\MyFirstApp\"

Назад

Создать

ФайлПравкаВидGitПроектСборкаОтладкаТестАнализСредстваРасширенияОкноСправкаПоискMyFirstApp

DebugAny CPUПуск

GitHub Copilot

Program.cs

MyFirstAppMyFirstApp.ProgramMain(string[] args)

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstApp
8 {
9     Ссылка: 0
10    internal class Program
11    {
12        Ссылка: 0
13        static void Main(string[] args)
14        {
15        }
16    }
17 }
```

100%Проблемы не найдены.

Стр: 1Симв: 1ПробелыCRLF

Вывод

Показать выходные данные из:

Обозреватель решений

Обозреватель решений — п

Решение "MyFirstApp" (1 про

MyFirstApp

Properties

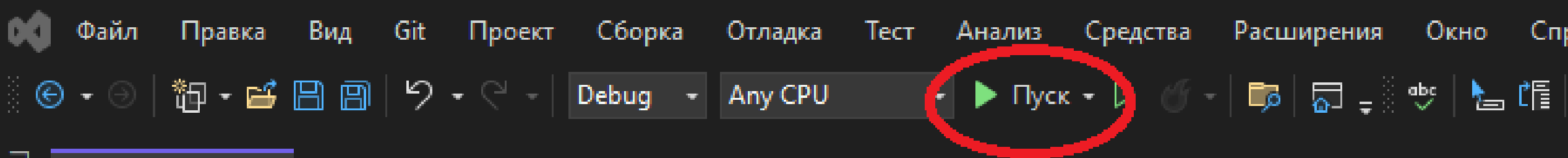
Ссылки

App.config

C# Program.cs

Обозрев...Измене...Предста...

Свойства



Панель элементов

Program.cs\* [icon] [X]

[C#] MyFirstApp

MyFirstApp.Program

```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Threading.Tasks;
6
7  namespace MyFirstApp
8  {
9      Ссылка: 0
9      internal class Program
10     {
11         Ссылка: 0
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Hello!");
14             Console.ReadKey();
15         }
16     }
17 }
```




C# MyFirstApp - MyFirstApp.Program - Main(string[] args)


100 %   Проблема

# Создание проекта

## Последние шаблоны проектов

 Консольное приложение (.NET Framework)

C#

 Консольное приложение (Майкрософт)

C#

Поиск шаблонов (ALT+"B")



Очистить все

C#

Все платформы

Все типы проектов



Консольное приложение (Майкрософт)

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

C#

Linux

macOS

Windows

Консоль



Blazor Web App

A project template for creating a Blazor web app that supports both server-side rendering and client interactivity. This template can be used for web apps with rich dynamic user interfaces (UIs).

C#

Linux

macOS

Windows

Blazor

Облако

Веб



Приложение .NET Aspire Starter (Майкрософт)

Шаблон проекта для создания приложения .NET Aspire с веб-интерфейсом Blazor и внутренней службой веб-API, при необходимости с использованием Redis для кэширования.

C#

.NET Aspire

API

Blazor

Облако

Common

Служба

Веб

Web API



Веб-приложение ASP.NET Core (Майкрософт)

Шаблон проекта для создания приложения ASP.NET Core с образцом содержимого ASP.NET Core Razor Pages

C#

Linux

macOS

Windows

Облако

Служба

Веб



Веб-API ASP.NET Core (Майкрософт)

Шаблон проекта для создания веб-API на основе REST с помощью контроллеров

Назад

Далее



# Настроить новый проект

Консольное приложение (Майкрософт)

C#

Linux

macOS

Windows

Консоль

Имя проекта

MyApp

Расположение

C:\Users\user\source\repos

Имя решения ⓘ

MyApp

☐ Поместить решение и проект в одном каталоге

Проект будет создан в "C:\Users\user\source\repos\MyApp\MyApp\"

Назад

Далее

## Дополнительные сведения

Консольное приложение (Майкрософт)

C#

Linux

macOS

Windows

Консоль

Платформа ⓘ

.NET 8.0 (долгосрочная поддержка)



Не использовать операторы верхнего уровня ⓘ



Включить публикацию native AOT ⓘ

Назад

Создать





## 2. Встроенные типы данных C#.

Целочисленные типы без знака (Unsigned):

- **byte**: Хранит целые числа от 0 до 255 (1 байт).
- **ushort**: Хранит целые числа от 0 до 65535 (2 байта).
- **uint**: Хранит целые числа от 0 до 4294967295 (4 байта).
- **ulong**: Хранит большие беззнаковые целые числа (8 байт).

Unsigned числа могут принимать только неотрицательные значения (от 0 и выше). Все биты используются для представления величины числа.

## Целочисленные типы со знаком (Signed):

- **sbyte**: Хранит целые числа от -128 до 127 (1 байт).
- **short**: Хранит целые числа от -32768 до 32767 (2 байта).
- **int**: Хранит целые числа от -2147483648 до 2147483647 (4 байта).
- **long**: Хранит целые числа в более широком диапазоне (8 байт).

Signed числа могут принимать как положительные, так и отрицательные значения. Для хранения знака числа используется **один бит**.

## Числа с плавающей точкой:

- **float:** Хранит числа с плавающей точкой одинарной точности (4 байта).
- **double:** Хранит числа с плавающей точкой двойной точности (8 байт).
- **decimal:** Хранит десятичные дроби с высокой точностью (16 байт), используется для финансовых расчетов.

## Логический тип:

**bool:** Хранит логические значения true или false.

## Символьный тип:

**char:** Хранит один символ Unicode (2 байта).

## Строковый тип:

**string:** Представляет последовательность символов Unicode.



## Логический тип

Имя типа в C#	Системный тип	Значения	Размер
<b>bool</b>	<b>Boolean</b>	true, false	8 бит

## Арифметические целочисленные типы

Имя типа	Системный тип	Диапазон	Размер
<b>sbyte</b>	<b>SByte</b>	-128 — 128	Знаковое, 8 Бит
<b>byte</b>	<b>Byte</b>	0 — 255	Беззнаковое, 8 Бит
<b>short</b>	<b>Short</b>	-32768 — 32767	Знаковое, 16 Бит
<b>ushort</b>	<b>UShort</b>	0 — 65535	Беззнаковое, 16 Бит
<b>int</b>	<b>Int32</b>	$\approx (-2 \cdot 10^9 \text{ — } 2 \cdot 10^9)$	Знаковое, 32 Бит
<b>uint</b>	<b>UInt32</b>	$\approx (0 \text{ — } 4 \cdot 10^9)$	Беззнаковое, 32 Бит
<b>long</b>	<b>Int64</b>	$\approx (-9 \cdot 10^{18} \text{ — } 9 \cdot 10^{18})$	Знаковое, 64 Бит
<b>ulong</b>	<b>UInt64</b>	$\approx (0 \text{ — } 18 \cdot 10^{18})$	Беззнаковое, 64 Бит

### Арифметический тип с плавающей точкой

Имя типа	Системный тип	Диапазон	Точность
<b>Float</b>	<b>Single</b>	$+1.5 \cdot 10^{-45} - +3.4 \cdot 10^{38}$	7 цифр
<b>double</b>	<b>Double</b>	$+5.0 \cdot 10^{-324} - +1.7 \cdot 10^{308}$	15-16 цифр

### Арифметический тип с фиксированной точкой

<b>decimal</b>	<b>Decimal</b>	$+1.0 \cdot 10^{-28} - +7.9 \cdot 10^{28}$	28-29 значащих цифр
----------------	----------------	--	---------------------

### Символьные типы

<b>char</b>	<b>Char</b>	U+0000 - U+ffff	16 бит Unicode символ
<b>string</b>	<b>String</b>	Строка из символов Unicode	

Минимальное и максимальное значения типа можно выяснить вызвав свойства **MinValue** и **MaxValue**:

```
Console.WriteLine("int: ");  
Console.WriteLine(int.MinValue);  
Console.WriteLine(int.MaxValue);  
Console.WriteLine("long: ");  
Console.WriteLine(long.MinValue);  
Console.WriteLine(long.MaxValue);  
Console.WriteLine("byte: ");  
Console.WriteLine(byte.MinValue);  
Console.WriteLine(byte.MaxValue);
```

C# Консоль отладки Microsoft V

```
int:  
-2147483648  
2147483647  
long:  
-9223372036854775808  
9223372036854775807  
byte:  
0  
255
```

**Метод GetType()** в C# является инструментом для получения информации о типе данных в процессе выполнения программы.

```
byte x = 5;  
Console.WriteLine(x.GetType());
```

```
int a = 26;  
int b = 0b11010;  
int c = 0x1A;  
  
Console.WriteLine(a);  
Console.WriteLine(b);  
Console.WriteLine(c);
```

Консоль отладки Microsoft Visual Studio

System.Byte

26

26

26

C:\Users\user\source\repos\myFirstApp\myFirstApp\Program.cs  
0.

## Точность различных числовых типов.

```
double value1 = 0.1;  
double value2 = 0.2;  
double result = value1 + value2;  
Console.WriteLine(result);
```

```
double resultRound = Math.Round(result, 2);  
Console.WriteLine(resultRound);
```

```
decimal value3 = 0.1m;  
decimal value4 = 0.2m;  
decimal result2 = value3 + value4;  
Console.WriteLine(result2);
```



Консоль отладки Microsoft Visual Studio

0,30000000000000000004

0,3

0,3

# Типы bool, char, string

```
bool isStudent = false;
bool isTeacher = true;

char simbol = 'X';

string hello = "hello";
string world = "World!";
string helloWorld = hello + world;
Console.WriteLine(helloWorld);

int age = 20;
Console.WriteLine("Антону " + age + " лет");
Console.WriteLine($"Антону {age} лет");
```

C:\> Консоль отладки Microsoft Vi

helloWorld!

Антону 20 лет

Антону 20 лет

C:\Users\user\source\rep  
0.

Чтобы автоматически закр  
томатически закрыть конс  
Нажмите любую клавишу, ч

## Приведение типов.

**Преобразование типов** в C# – это процесс изменения типа данных одной переменной на другой.

Существует два основных вида преобразований:

**Неявные преобразования:** Происходят автоматически, когда компилятор может выполнить преобразование без потери данных. Например, преобразование `int` в `double`, так как число с плавающей запятой может вместить любое целое число.

**Явные преобразования:** Требуют указания типа в скобках. Используются, когда существует риск потери данных или когда преобразование между несовместимыми типами. Например, преобразование `double` в `int` может привести к потере дробной части.

# Неявные преобразования

Расширяющие преобразования: Происходят, когда целевой тип имеет больший диапазон значений, чем исходный.

```
int valueInt = 33;  
// неявное приведение int к double  
double valueDouble = (double)valueInt;
```

```
byte nuber = 222;  
// неявное приведение byte к int  
int mnumber2 = nuber;
```



С дт.	По
sbyte	short, int, long, float, double, decimal или nint
byte	short, ushort, int, uint, long, ulong, float, double, decimal, nint или nuint
short	int, long, float, double или decimal либо nint
ushort	int, uint, long, ulong, float, double или decimal, nint или nuint
int	long, float, double или decimal, nint
uint	long, ulong, float, double или decimal либо nuint
long	float, double или decimal
ulong	float, double или decimal
float	double

## Явные преобразования (приведение)

Сужающие преобразования: Происходят, когда целевой тип имеет меньший диапазон значений, чем исходный. **Могут привести к потере данных.**

double -> int (дробная часть отбрасывается)

long -> int (может произойти переполнение)

**Синтаксис: (тип\_целевой\_переменной) значение**

```
double number = 10.0;  
int integer = (int)number; // Явное приведение к типу int
```

```
double number = 10.5;  
// Явное приведение к типу int  
int integer = (int)number;  
  
int asciiCode = 65;  
// Явное приведение к типу char  
char character = (char)asciiCode;  
  
long largeNumber = 100000;  
// Явное приведение от long к int  
int smallerNumber = (int)largeNumber;
```

С дт.	По
sbyte	byte, ushort, uint, ulong или nuint
byte	sbyte
short	sbyte, byte, ushort, uint, ulong или nuint
ushort	sbyte, byte или short
int	sbyte, byte, short, ushort, uint, ulong или nuint
uint	sbyte, byte, short, ushort, int или nint
long	sbyte, byte, short, ushort, int, uint, ulong, nint или nuint
ulong	sbyte, byte, short, ushort, int, uint, long, nint или nuint
float	sbyte, byte, short, ushort, int, uint, long, ulong, decimal, nint или nuint
double	sbyte, byte, short, ushort, int, uint, long, ulong, float, decimal, nint или nuint

# Контроль переполнения

**checked** - это ключевое слово в C#, которое используется для включения проверки на переполнение при арифметических операциях и преобразованиях типов.

Когда вы помещаете код внутри блока `checked`, компилятор будет генерировать код для проверки на переполнение, и если оно происходит, будет выброшено исключение `OverflowException`.

```
checked
```

```
{
```

```
    int count = 55;
```

```
    byte count2 = (byte)count; // Явное преобразование: int -> byte, без потерь
```

```
    Console.WriteLine(count2);
```

```
}
```

Консоль отладки Microsoft Visual Studio

55

```
checked
```

```
{
```

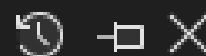
```
    int count = 555;
```

```
    byte count2 = (byte)count; // Явное преобразование: int -> byte, с потерями ❌
```

```
    Console.WriteLine(count2);
```

```
}
```

Исключение не обработано



**System.OverflowException:** "Arithmetic operation resulted in an overflow."

# 3. Переменные. Константы.

**Переменная** — это именованная область памяти, предназначенная для хранения данных определенного типа.

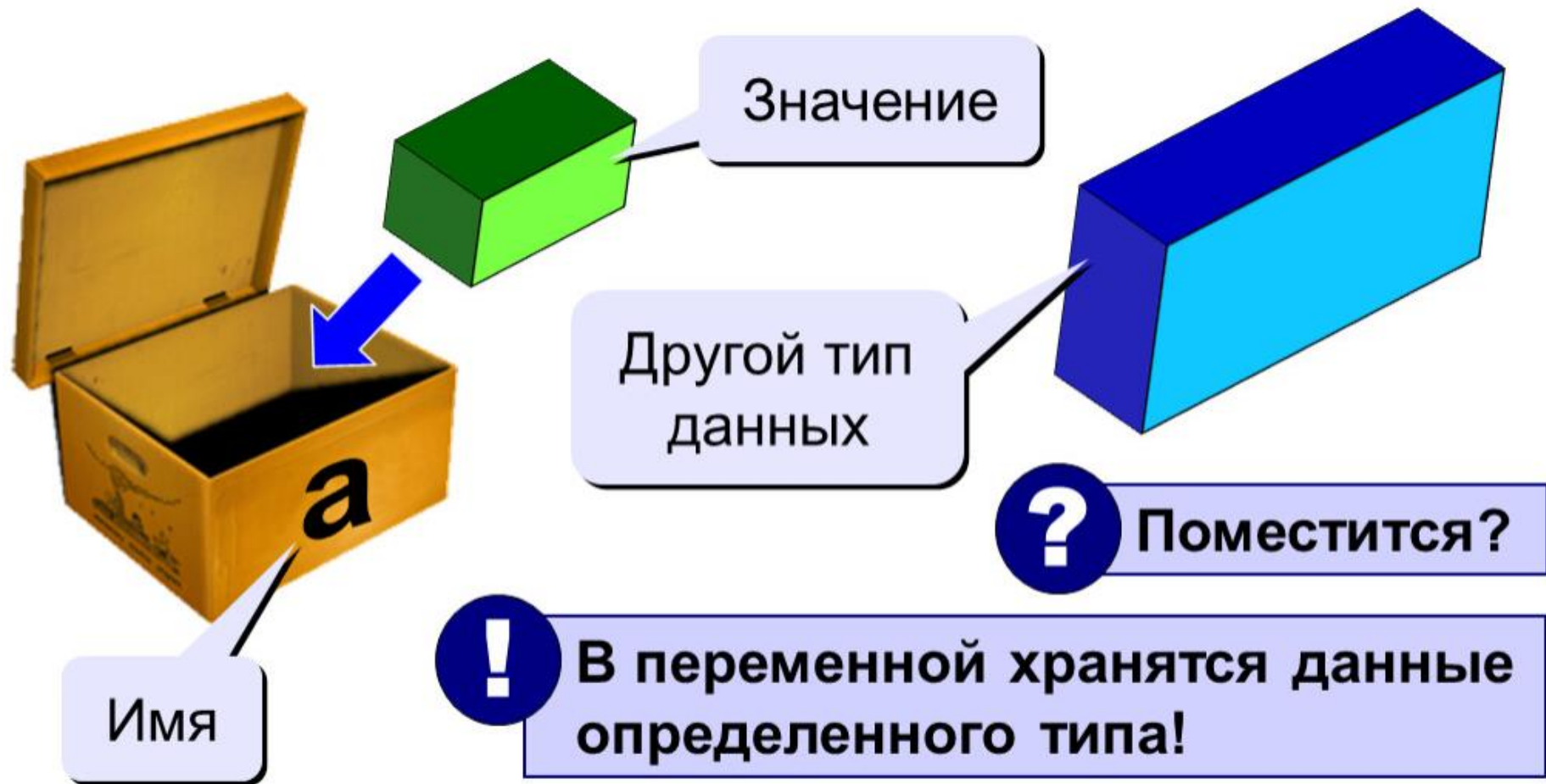
**Ключевые характеристики переменной:**

**Имя:** Уникальное имя, по которому вы обращаетесь к переменной в программе.

**Тип:** Определяет, какие значения может хранить переменная (например, целые числа, числа с плавающей запятой, строки, логические значения).

**Значение:** Конкретные данные, хранящиеся в переменной в данный момент времени.

**Переменная** — это величина, имеющая имя, тип и значение. Значение переменной можно изменять во время работы программы.





# Объявление переменных

```
// Синтаксис объявления переменной: тип имя;  
// Точка с запятой (;) отделяет друг от друга отдельные инструкции (операторы).  
int value; // выделить память компьютера (4 байта), назначить имя "value"  
  
// Инициализация переменной. (Использовать переменную можно только после инициализации)  
// имя = значение;  
// "=" – оператор присваивания  
value = 1; // записать в ячейку памяти с именем "value" значение "1"  
  
// Синтаксис объявления переменной с одновременной инициализацией: тип имя = значение;  
double pi = 3.14; // выделить память компьютера (8 байт), назначить имя "pi", записать в ячейку памяти "3.14"  
  
value = 33; // значение переменной можно изменять  
  
byte value = 200; // нельзя объявить две переменные с одинаковым именем в одной области видимости
```

## Основные правила именования переменных:

**Первый символ:** Имя переменной должно начинаться с буквы (заглавной или строчной), подчеркивания (`_`) или символа `@`.

**Последующие символы:** После первого символа могут использоваться буквы, цифры и подчеркивания.

**Регистр:** C# чувствителен к регистру, то есть переменные `name` и `Name` считаются разными.

**Зарезервированные слова:** Нельзя использовать в качестве имен переменных ключевые слова языка C# (например, `int`, `double`, `if`, `else` и т.д.).

# Ключевые слова C#

break	decimal	double	float	fixed
checked	else	extern	implicit	if
default	false	for	is	internal
enum	foreach	in	null	new
finally	int	lock	params	override
goto	long	object	ref	readonly
interface	operator	private	sizeof	short
namespace	protected	return	switch	struct
out	sbyte	stackalloc	typeof	try
public	static	this	ushort	unsafe
sealed	throw	uint	while	volatile
string	ulong	using as	abstract	
true	virtual	case	byte	
unchecked	base	const	class	
void.bool	catch	do	delegate	
char	continue	explicit	event	

**Верблюжья нотация** (или camelCase) – это соглашение об именовании идентификаторов в программировании, при котором первое слово пишется со строчной буквы, а каждое последующее слово – с заглавной.

```
// Примеры именования переменных  
int age;  
float price;  
string firstName;  
string secondName;  
byte countOfStudents;  
int a, b, c; // допустимо для математических выражений
```

**Константа** в C# - это переменная, значение которой устанавливается один раз при объявлении и не может быть изменено впоследствии в ходе выполнения программы.

## **Зачем использовать константы?**

- **Повышение читаемости кода:** Замена числовых литералов на осмысленные имена констант делает код более понятным и поддерживаемым.
- **Уменьшение ошибок:** Если какое-то значение используется в нескольких местах программы, изменение его значения в одном месте может привести к ошибкам. Использование константы позволяет избежать таких ошибок, так как значение изменяется только в одном месте.
- **Улучшение производительности:** Компилятор может оптимизировать код, если знает, что значение переменной не изменяется.

# Объявление констант

Константы объявляются с помощью ключевого слова **const** перед типом данных:

```
const double pi = 3.1415;  
const int maxAge = 120;  
const double gravity = 9.81;
```

## 4. Комментарии.

**Комментарии** в программировании — это специальные строки текста, которые игнорируются компилятором или интерпретатором при выполнении программы. Они служат для пояснения кода и облегчения его понимания как для самого программиста, так и для других разработчиков, которые могут работать с этим кодом в будущем.

```
int x = 10; // однострочный комментарий
```

```
/*  
    многострочный комментарий  
*/
```

# 5. Литералы.

Литералы в C# представляют собой фиксированные значения, которые напрямую используются в коде.

Литералы могут быть различных типов: числовые, строковые, символьные и т.д.

Рассмотрим основные типы литералов в C#.



В языке программировании C# для явного указания типа числового литерала используются **суффиксы**. Это позволяет избежать неявных преобразований типов и повысить точность вычислений.

## **Суффиксы целочисленных типов.**

C# позволяет использовать суффиксы для явного указания типа целочисленного литерала:

**U** или **u** - uint (беззнаковое целое)

**L** или **l** - long (целое с плавающей запятой)

**UL** или **ul** - ulong (беззнаковое целое с плавающей запятой)

Примеры:

```
int number1 = 10; // Тип int по умолчанию  
uint number2 = 10U; // Тип uint  
long number3 = 10L; // Тип long  
ulong number4 = 10UL; // Тип ulong
```

Суффиксы целочисленных типов не обязательны при инициализации переменных в C#.

Компилятор C# пытается определить тип литерала автоматически. Если вы не используете суффикс, компилятор предполагает, что литерал имеет тип **int**, если его значение не выходит за пределы диапазона **int**. Использование суффиксов может повысить читаемость кода, делая явным тип данных, с которым вы работаете.

```
int number1 = 10; // Тип int по умолчанию
```

```
// Тип long, так как значение слишком большое для int  
long number2 = 1000000000000L;
```

Префикс **0b** указывает, что литерал записан в двоичной системе счисления.

Например: **0b1011** равно десятичному числу 11.

Префикс **0x** указывает, что литерал записан в шестнадцатеричной системе счисления.

Например: **0x2A** равно десятичному числу 42.

```
int binaryNumber = 0b1011; // Двоичное число 1011
int hexadecimalNumber = 0x2A; // Шестнадцатеричное число 2A

Console.WriteLine(binaryNumber); // Вывод: 11
Console.WriteLine(hexadecimalNumber); // Вывод: 42
```

## Примеры:

```
int decimalNumber = 42;           // десятичное число
int hexadecimalNumber = 0x2A;     // 16-ричное число
int binaryNumber = 0b1010;       // двоичное число
uint unsignedNumber = 123U;       // беззнаковое десятичное число
long longNumber = 1234567890L;    // 10-е число типа Long
```

## Суффиксы для чисел с плавающей точкой:

**f** или **F**: Определяет тип **float**.

**d** или **D**: Определяет тип **double**.

**m** или **M**: Определяет тип **decimal**.

```
float floatNumber = 10.5f; // Тип float  
double doubleNumber = 10.5d; // Тип double  
decimal decimalNumber = 10.5m; // Тип decimal
```

Тип **double** является типом по умолчанию для чисел с плавающей точкой.

# Использование суффиксов для явного указания типа числа.

```
Console.WriteLine("Без явного указания типа");  
float value = 4 / 3;  
Console.WriteLine(value);
```

```
double value2 = 4.0 / 3.0;  
Console.WriteLine(value2);
```

```
Console.WriteLine("С явным указанием типа");  
float value3 = 4f / 3f;  
Console.WriteLine(value3);
```

```
double value4 = 4d / 3d;  
Console.WriteLine(value4);
```

cs Консоль отладки Microsoft Visual Studio

Без явного указания типа

1

1,3333333333333333

С явным указанием типа

1,3333334

1,3333333333333333

C:\Users\user\source\repos\myFirstApp\myFirstApp\Program.cs  
0.

Чтобы автоматически закрывать консоль при завершении программы, добавьте в файл Program.cs следующий код:

**Экспоненциальное** представление чисел с плавающей точкой используется для записи очень больших или очень маленьких чисел в более компактном виде.

Синтаксис: **<мантисса>е<экспонента>**

**Мантисса:** Это десятичное число, которое представляет основную часть числа.

**е:** Это символ, который отделяет мантиссу от экспоненты.

**Экспонента:** Это целое число, которое указывает степень десяти, на которую умножается мантисса.



## Примеры:

```
double number1 = 1.23e+4; // 12300 (1.23 * 10^4)
double number2 = 5.67e-3; // 0.00567 (5.67 * 10^-3)
double number3 = 1e+9; // 1000000000 (1 * 10^9)
```

```
float numberFloat = 3.14f;
double numberDouble = 1.23e-5;
double numberDefault = 404.404;
decimal decimalNumber = 123.45m;
```

## Подчеркивание для улучшения читаемости

С C# 7 можно использовать символ подчеркивания (\_) для разделения разрядов в числовых литералах, что улучшает читаемость длинных чисел:

```
int largeNumber = 1_000_000;  
double pi = 3.14159_26535;
```

## Важные моменты

- Выбор типа: Выбирайте подходящий числовой тип в зависимости от диапазона значений и требуемой точности.
- Переполнение: Присвоение слишком большого значения переменной меньшего типа может привести к переполнению.
- Точность: Типы с плавающей точкой имеют ограниченную точность, особенно при представлении дробных чисел.
- Decimal: Тип `decimal` используется для точных денежных расчетов, так как имеет высокую точность и не подвержен ошибкам округления.

**Булевы литералы** (или логические литералы) представляют собой константные значения, которые могут принимать только одно из двух возможных значений: **true** (истина) или **false** (ложь).

Они используются для представления логических условий и результатов сравнений.

```
bool isTrue = true;  
bool isFalse = false;
```

## Важные моменты

- Тип данных: Булевы значения имеют тип `bool`.
- Неявное преобразование: Булевы значения не могут быть неявно преобразованы в другие типы данных.
- Логические операции: К булевым значениям можно применять логические операции: `&&` (И), `||` (ИЛИ), `!` (НЕ).

# Символьные литералы.

Символьные литералы заключаются в одинарные кавычки (' ').

Тип данных: - **char**.

Примеры:

'A' - символ "A"

'1' - символ "1"

```
char a = 'a';  
char simbol_B = 'B';
```

## Строчные литералы.

Строчные литералы заключаются в двойные кавычки (" ").

Тип данных: - **string**.

```
string path = "C:\\Users\\Public\\Documents"; // Путь к файлу
string message = "Hello, world!";
string welcome = "Добро пожаловать в отель \"Калифорния\"!";
```

## Escape-последовательности.

Escape-последовательности используются для представления специальных символов, которые нельзя ввести напрямую.

Они начинаются с обратного слеша (\).

### Примеры:

\n - новая строка

\t - табуляция

\\ - обратный слеш

\\" - двойная кавычка

\' - одинарная кавычка



**Дословный строковый литерал** (префиксом @) - это способ представления строки, в которой все символы, включая специальные символы, интерпретируются буквально, без использования escape-последовательностей.

```
string path = @"C:\Users\Public\Documents"; // Путь к файлу
string message = @"Hello, world!";
string welcome = @"Добро пожаловать в отель ""Калифорния""!";
```

# **null**

Литерал **null** представляет собой отсутствие значения. Он используется для обозначения того, что переменная не содержит никакой информации или не ссылается на объект.

**null** и **0** - это совершенно разные понятия.

## **Домашнее задание:**

1. Повторить материал лекции.
2. Установить на домашнем ПК Visual Studio, проверить работоспособность программы.

# **Материалы лекций:**

<https://github.com/ShViktor72/Education>

# **Обратная связь:**

[colledge20education23@gmail.com](mailto:colledge20education23@gmail.com)

# Список литературы:

1. Жаксыбаева Н.Н. Основы объектно-ориентированного программирования: язык C#. Часть 1./ Учебное пособие предназначено для учащихся технического и профессионального образования, Алматы, 2010,
2. <https://learn.microsoft.com/ru-ru/dotnet/csharp/>