

ЛЕКЦИЯ 2

- 06.09.2023
- Платформа .NET
- Система типов данных языка C#



**Почему
вы стали
учителем?**



**Понимаешь... Если
разговаривать с собой наедине,
это шизофрения. А вот если
разговаривать с собой в
присутствии учеников - это
лекция.**

ЦЕЛИ ЛЕКЦИИ

- Получить основные сведения о платформе .NET
- Познакомится с системой типов языка C#
- Научиться описывать и инициализировать переменные
- Разобраться с приведением и преобразованием типов



Это изображение, автор: Неизвестный автор, лицензия: CC BY-NC

ПЛАТФОРМА

- **Платформа** [platform] – наиболее общее описание программно-аппаратной (в том числе сетевой) среды, на которой разрабатывается и/или развёртывается прикладное ПО. Обязательно включает архитектуру компьютера, тип ОС и средства доступа к данным (СУБД).
 - Платформа разработки [development platform]
 - Платформа развёртывания [deployment platform]: средства разработки ПО и необходимое программное окружение готового ПО соответственно

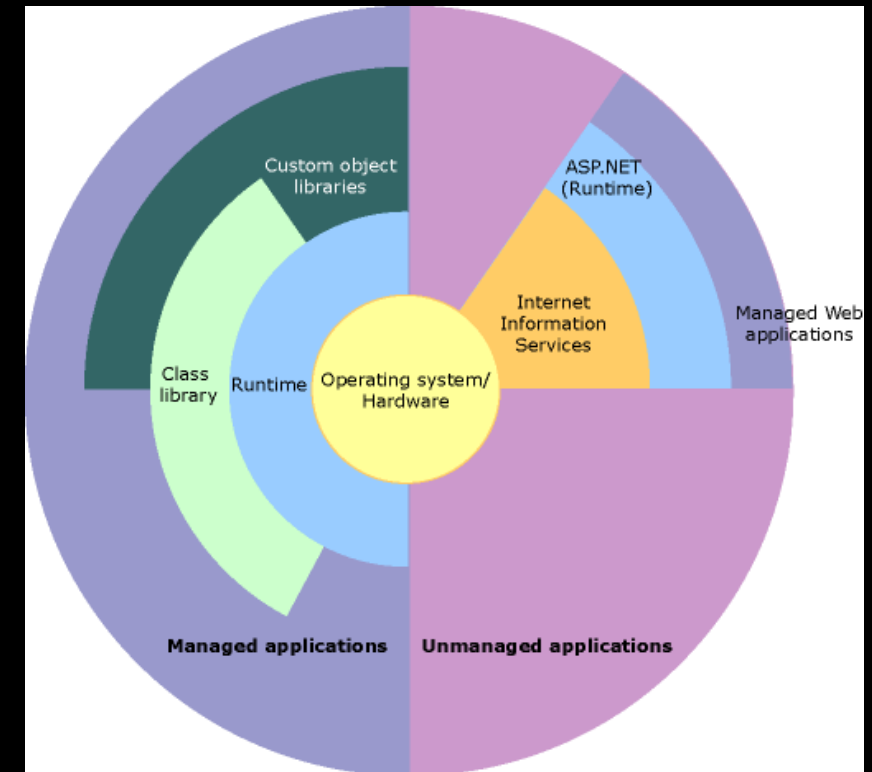
ПЛАТФОРМА .NET

- **.NET** - open source платформа для создания кроссплатформенных приложений
 - Открытый код на github
[<http://github.com/dotnet/core>]
- Общезыковая среда выполнения (Common Language Runtime, CLR)
- Библиотека классов .NET Framework

.NET Glossary [<https://docs.microsoft.com/en-us/dotnet/standard/glossary>]

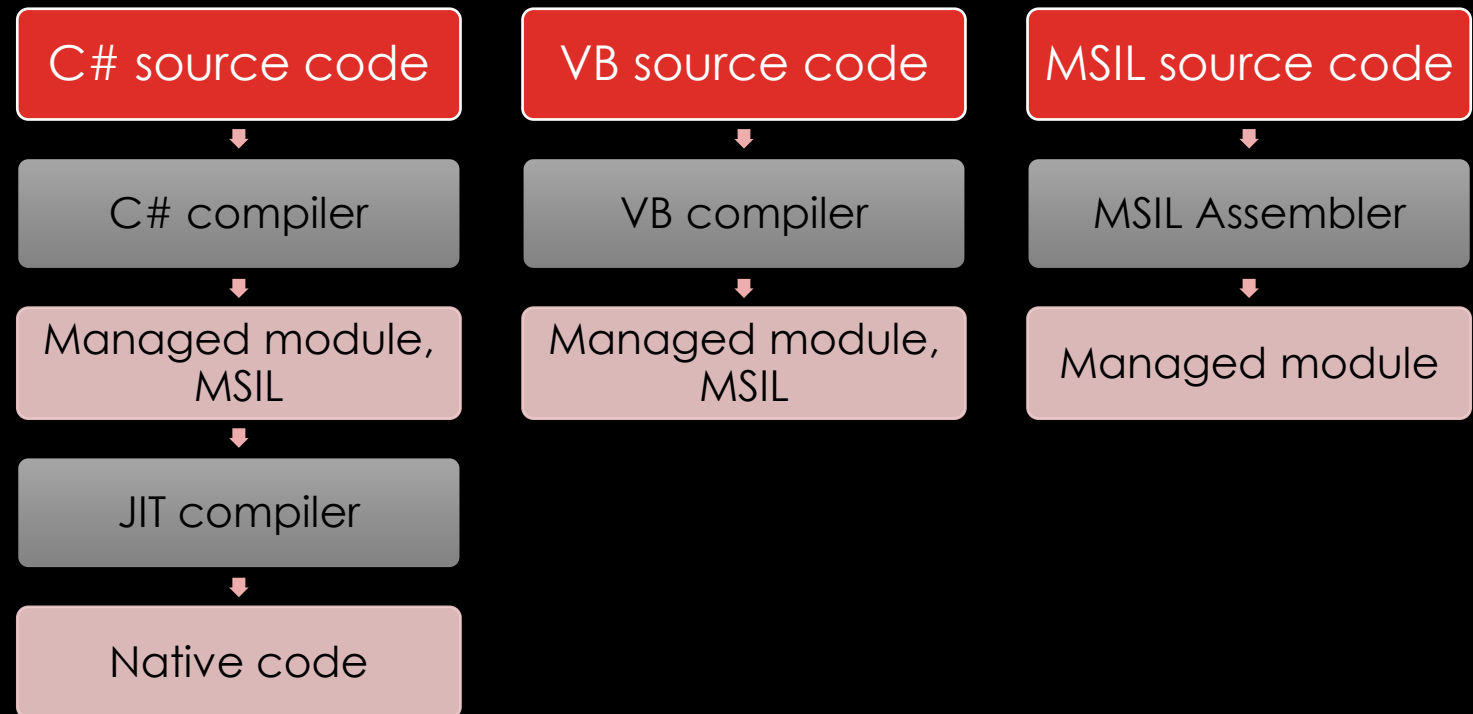
.NET implementations [<https://docs.microsoft.com/en-us/dotnet/fundamentals/implementations#applicable-standards>]

Общие сведения о платформе .NET [<http://docs.microsoft.com/ru-ru/dotnet/framework/get-started/overview>]
Максименкова О.В., 2023



ОБЩЕЯЗЫКОВАЯ СРЕДА ИСПОЛНЕНИЯ CLR (1)

- **Управляемый код** (в понимании .NET, конечно) – это код, выполнение которого управляется средой выполнения (CLR)



ОБЩЕЯЗЫКОВАЯ СРЕДА ИСПОЛНЕНИЯ CLR (2)

Общезыковая среда исполнения CLR управляет

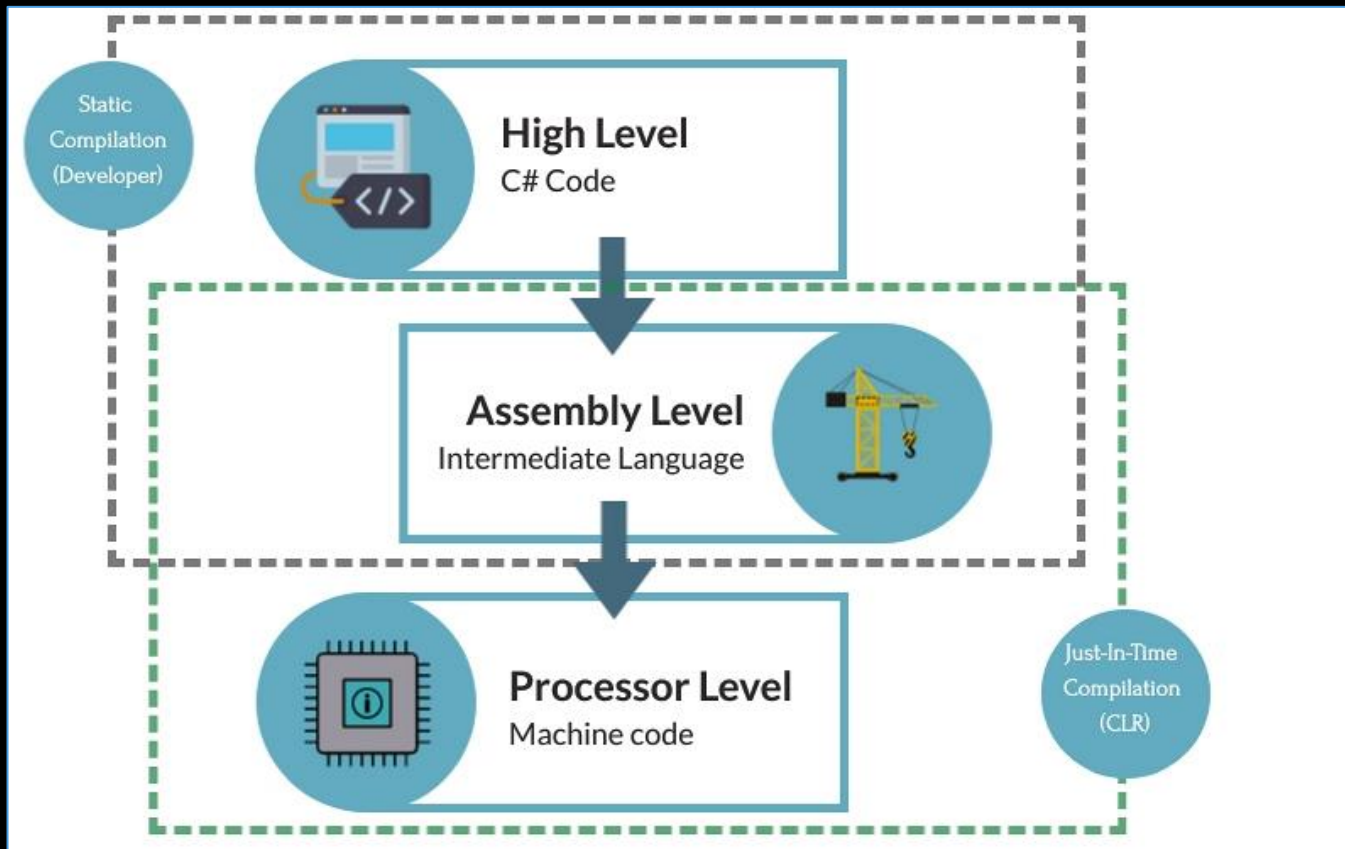
- памятью и потоками выполнения кода
- проверкой безопасности кода
- компиляцией в машинный код
- выполнением машинного кода
- ...

Плюсы от использования CLR

- Повышение производительности
- Возможность и упрощение использование компонентов, разработанных на разных ЯП
- Сборка мусора
- Делегаты вместо указателей на функцию повышают типобезопасность и надёжность
- Поддержка упорядоченного механизма обработки исключений
- Поддержка модели свободных потоков
- Поддержка кастомных атрибутов

Common Language Runtime (CLR) overview [<http://docs.microsoft.com/en-us/dotnet/standard/clr>]

ПРОЦЕСС КОМПИЛЯЦИИ С#-КОДА



Промежуточный язык
Microsoft Intermediate language, MSIL
или
Common Intermediate Language
или
Intermediate Language, IL

JIT-компилятор
JIT-компиляция происходит во время
исполнения приложения, на той же
машине, где выполняется код

РАЗРАБОТЧИК:

НА МОЕМ КОМПЬЮТЕРЕ
ВСЕ РАБОТАЕТ



ПРОДАКТ-МЕНЕДЖЕР:

ДА, НО МЫ НЕ СОБИРАЕМСЯ
ОТДАВАТЬ ТВОЙ КОМПЬЮТЕР
ЗАКАЗЧИКУ



СБОРКА (.EXE ИЛИ .DLL)

Манифест
(Обязателен)

Управляемый модуль 1

IL-Код +
метаданные



Ресурсы
(html, gif,...)

.....

Управляемый модуль N

IL-Код +
метаданные



Ресурсы
(html, gif,...)

Характеристики сборки

- Сборка определяет повторно используемые типы
- Сборка связана с номером версии
- Со сборкой может быть ассоциирована информация о безопасности

сборка – это единица повторного использования, версионирования и безопасности

Д. Рихтер CLR via C#

Роль сборок .NET

[http://professorweb.ru/my/csharp/assembly/level1/1_2.php]

СИСТЕМА ОБЩИХ ТИПОВ

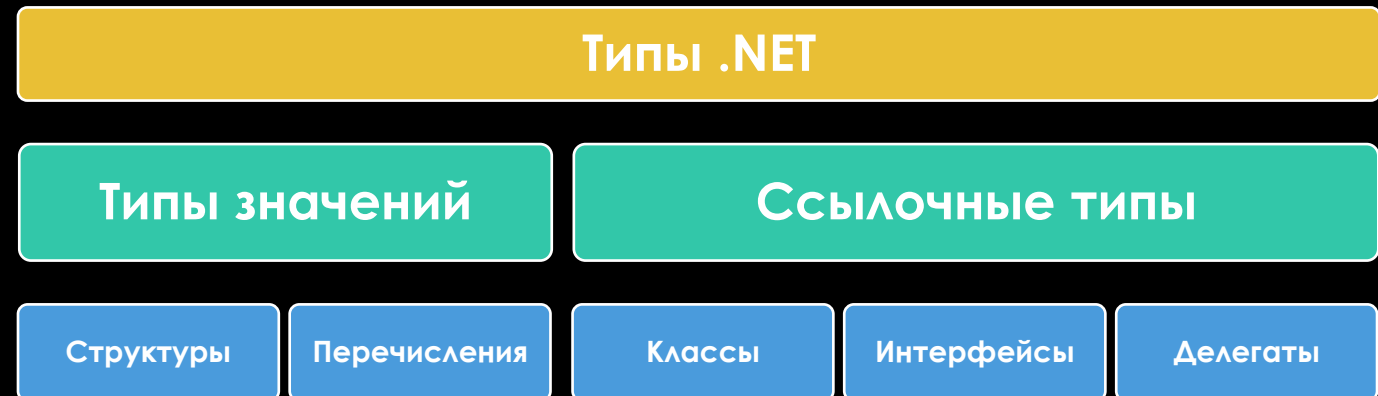
Common Type System, CTS – система, определяющая способ объявления, использования и управления типами в CLR

Функции CTS

- Формирует инфраструктуру, которая позволяет обеспечивать межязыковую интеграцию, безопасность типов и высокопроизводительное выполнение кода
- Предоставляет объектно-ориентированную модель, поддерживающую полную реализацию многих языков программирования
- Определяет правила, которых необходимо придерживаться в языке обеспечения взаимодействия объектов, написанных на разных языках
- Предоставляет библиотеку, которая содержит простые (элементарные) типы, такие как Boolean, Byte, Char, Int32 и UInt64

Максименкова О.В., 2023

Категории типов в CTS



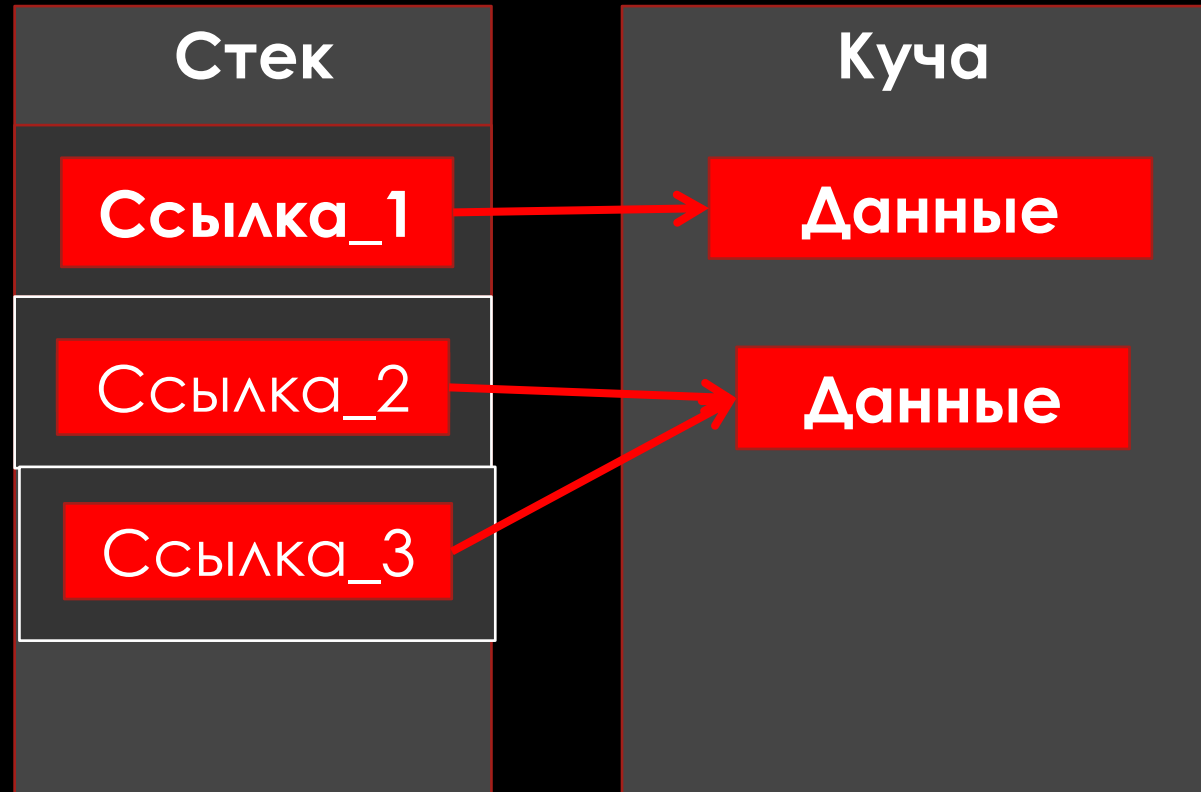
- **Типы значений** — это типы данных, объекты которых представлены фактическим значением объекта.
- **Ссылочные типы** — это типы данных, объекты которых представлены ссылкой (аналогичной указателю) на фактическое значение объекта.

РАЗМЕЩЕНИЕ ДАННЫХ В ПАМЯТИ

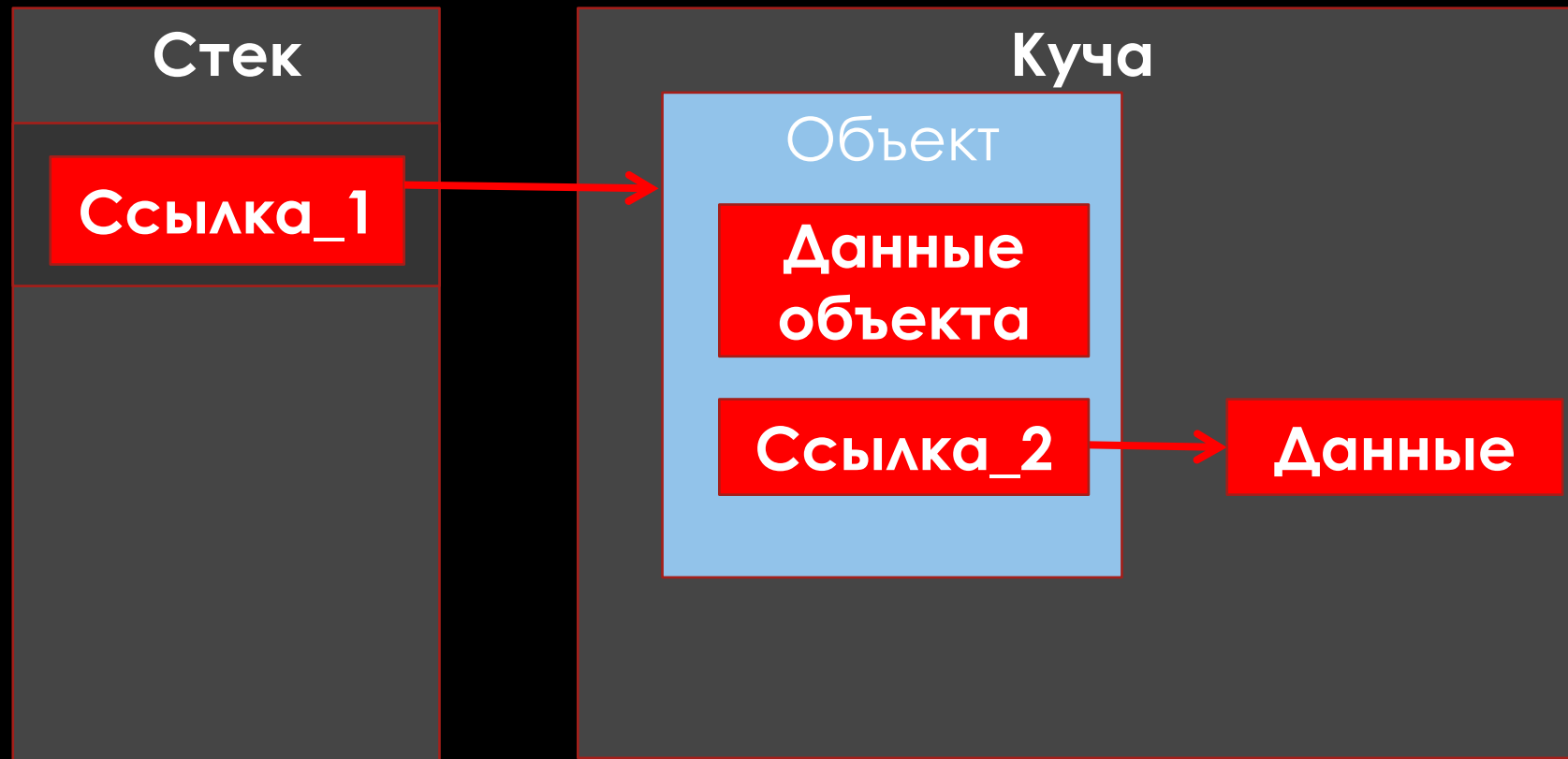
Данные с типами значений вне объектов



Данные ссылочных типов вне объектов



ДАННЫЕ ССЫЛОЧНОГО ТИПА В ОБЪЕКТЕ



СИСТЕМА ТИПОВ ЯЗЫКА С#

Данные и типы данных

Чтение данных

Обработка исключений при работе с данными



ТИП ДАННЫХ И СИСТЕМА ТИПОВ

- **Тип данных** (конкретный тип) [data type] – характеристика, явно или неявно приписываемая объекту (переменной, константе, полю записи, выражению, функции и т. п.) и определяющая множество допустимых значений, формат хранения данных, размер выделяемой под них памяти и набор операций, которые над ними можно производить
- Главная характеристика типа данных – **множество допустимых значений** (МДЗ). Это отличает его от абстрактного типа данных (АТД).
- **Мощность типа** – мощность его МДЗ

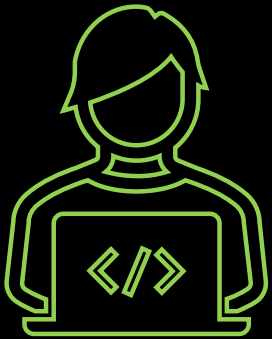
СИСТЕМА ТИПОВ ЯП И ЕЕ ХАРАКТЕРИСТИКИ

- Система типов [type system] языка программирования определяет состав базовых типов языка, способы создания новых типов и правила их использования
- Контроль соответствия типов [type checking] – проверка того, что тип переменной совместим с типом выражения при выполнении оператора присваивания, при подстановке аргумента на место формального параметра и в других случаях сопоставления различных данных между собой

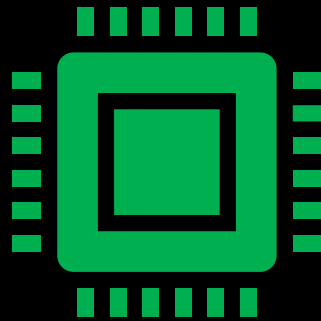
Чем строже контроль типов, тем меньше дополнительных соглашений о совместимости типов в языке программирования

ПОДХОДЫ К КОНТРОЛЮ СООТВЕТСТВИЯ ТИПОВ

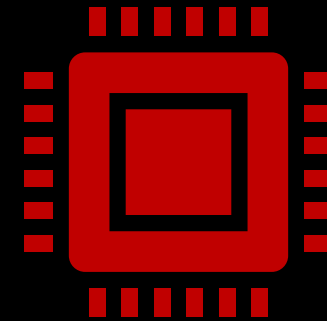
Кто осуществляет контроль соответствия типов?



Программист



Компилятор выполняет неявное
преобразование типов данных



Компилятор выполняет строгий
контроль соответствия типов и
запрещает действия,
противоречащие соглашениям
о совместимости типов данных

СВОЙСТВА СИСТЕМЫ ТИПОВ

По времени назначения типа
некоторой переменной

- Статическая типизация
- Динамическая типизация

По строгости контроля типов

- Строгая типизация
- Слабая типизация

СВОЙСТВА СИСТЕМЫ ТИПОВ C# (1)

Строгая типизация

Элемент данных x
имеет тип int

```
int x = 15;
```

```
var b = 33;  
Console.WriteLine(b.GetType());
```

Тип данных для элемента
b выводится по типу
значения справа

```
b = "ABD"; // Ошибка компиляции.
```

СВОЙСТВА СИСТЕМЫ ТИПОВ C# (2)

Статическая типизация

```
int x = 15;
```

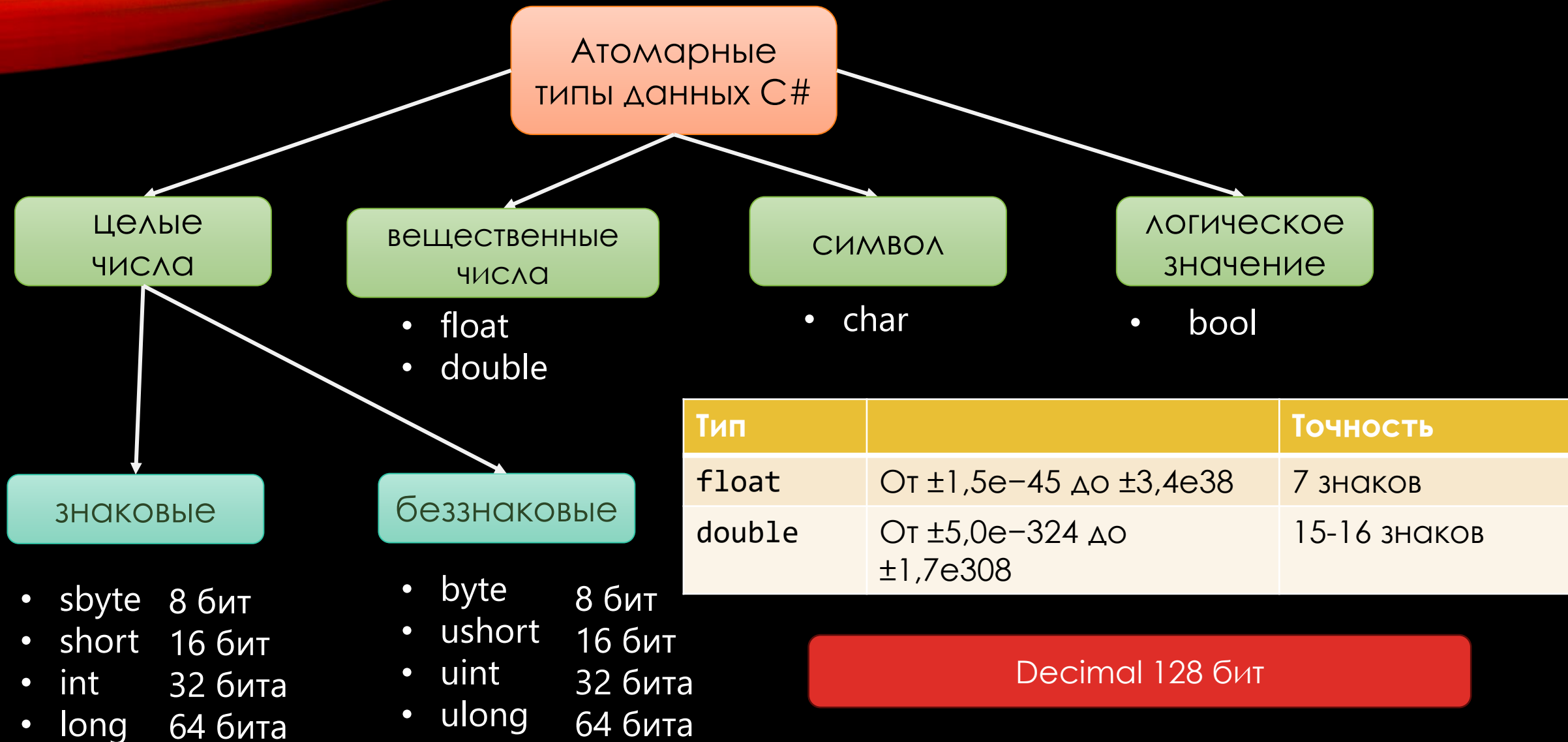
```
double x = 3.14; // Ошибка компиляции.
```

Переменная связана с типом `int` до конца своей жизни в этой области видимости программы

Поэтому ей нельзя сменить тип данных просто так

ВИДЫ ТИПОВ ДАННЫХ

- **Атомарные (простые) типы** описывают типы атомарных информационных элементов на физическом уровне.
 - Например: целое число, вещественное число, символ, логическое значение
- **Составные (комплексные) типы** – типы данных, образованные путём объединения в некоторую структуру набора простых типов
 - Базовым способом объединения данных является агрегирование
- **Кортеж [tuple]** – упорядоченный набор из n элементов (где n – натуральное число) произвольной природы, называемых компонентами или координатами



ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В ПАМЯТИ КОМПЬЮТЕРА

- Целые
 - Беззнаковые (представлены в прямом коде)
 - Знаковые (представлены в дополнительном коде)
- Вещественные
 - Стандарт представления вещественных чисел IEEE 754

Таблица типов с плавающей запятой (Справочник по C#) [[https://docs.microsoft.com/ru-ru/previous-versions/visualstudio/visual-studio-2008/9a9e1949\(v=vs.90\)?redirectedfrom=MSDN](https://docs.microsoft.com/ru-ru/previous-versions/visualstudio/visual-studio-2008/9a9e1949(v=vs.90)?redirectedfrom=MSDN)]

IEEE 754 Converter [<http://www.h-schmidt.net/FloatConverter/IEEE754.html>]

ПЕРЕМЕННЫЕ

Идентификаторы
Объявление и инициализация



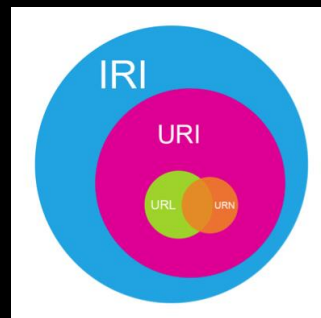
ИДЕНТИФИКАТОРЫ

Идентификатор [identifier] – имя сущности, обладающее следующими свойствами:

- каждая сущность имеет только один идентификатор
- различные сущности не могут иметь совпадающие идентификаторы
- время жизни сущности совпадает со временем жизни идентификатора



Это изображение, автор:
Неизвестный автор, лицензия: [CC BY](#)



Это изображение, автор: Неизвестный автор, лицензия: [CC BY-NC-ND](#)



Это изображение, автор: Неизвестный автор, лицензия: [CC BY](#)



ИДЕНТИФИКАТОРЫ C#

- **не могут** начинаться с цифр или содержать пробелов
- **могут** содержать буквенные символы Юникода, десятичные числа, символы соединения Юникода, несамостоятельные знаки Юникода или символы форматирования Юникода
- **не может** содержать символов @ и \$ (@ допустим как первый символ для получения буквального идентификатора)

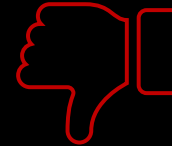


Корректно:

@if
_007
Якорь
S_2_4_6
День_недели

Некорректно:

if
666
номер дома
name@
mail.ru



ОБЛАСТЬ ВИДИМОСТИ И ДЕЙСТВИЯ ИДЕНТИФИКАТОРА

- Область видимости или область действия [scope, visibility scope] идентификатора – участок текста программы, в котором этот идентификатор можно использовать
 - В различных областях видимости могут встречаться одинаковые идентификаторы, которые будут обозначать различные сущности

Уровень класса

Уровень метода

Уровень блока кода

ОБЛАСТИ ВИДИМОСТИ ИДЕНТИФИКАТОРОВ С#

1

```
{  
    int x = 10;  
    Console.WriteLine(x);  
}  
Console.WriteLine(x);
```

2

```
{  
    int x = 10;  
}  
double x = 15;
```

Контекст / Context и Область видимости (действия) / Scope – это не одно и то же!

ОФОРМЛЕНИЕ КОДА C#

- **Идентификаторы, обозначающие имена методов и типов**

- PascalCase

Примеры: `ReadLine()`, `Console`, `Math`, `WriteLine()`...

- **Идентификаторы, обозначающие локальные переменные**

- camelCase
- существительные на английском языке, отражающие содержимое переменной
- Множественное число используется для именования переменных коллекций
 - Примеры: `number`, `firstDigit`...

ПЕРЕМЕННЫЕ

- **Переменная** [variable] – именованный элемент данных, значение которого может изменяться
- **Имя** [name] **переменной** – правильный идентификатор языка программирования, однозначно определяющий переменную
 - Переменная содержит некоторое значение. Само по себе «значение» – базовое неопределяемое понятие
- **Значение** [value] **переменной** – значение элемента данных, который идентифицируется именем переменной
- **Тип** [type] **переменной** – тип данных, которые могут содержаться в переменной
- **Адрес** [address] **переменной** – адрес начала участка памяти, занимаемого значением переменной
- **Объявление** [declaration] **переменной** – первое упоминание переменной, задающее имя, тип и другие свойства переменной

КАТЕГОРИИ ПЕРЕМЕННЫХ В C#

Название	Член типа	Описание (назначение)
Локальная переменная метода	Нет	Для временного размещения данных внутри области видимости метода или другого функционального члена
Поле	Да	Для данных, ассоциированных с типом
Параметр метода	Нет	Для временных переменных, используемых для передачи данных из одного метода в другой
Элемент массива	Да	Для временных данных или данных, ассоциированных с типом

ОБЪЯВЛЕНИЕ И ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ В C#

- **Объявление (создание) переменной** типа `Type`:
 - `Type имя_переменной`;
`double x;`
- **Присваивание существующей переменной значения**:
 - `имя_переменной = выражение`;
`x = 7.5;`
- **Объявление и инициализация**:
 - `Type имя_переменной = выражение`;
`int bin = 2, oct = 8, hex = 16, testVal = 1024;`

Описать переменную желательно максимально близко к месту первого использования в коде программы

КАТЕГОРИИ ПЕРЕМЕННЫХ В C#

Название	Размещение	Инициализация	Назначение (использование)
Локальная переменная метода	Стек или куча	Нет	Для локальных вычислений в функциональном члене
Поле класса	Куча	Есть	Член класса
Поле структуры	Стек или куча	Есть	Член структуры
Параметр метода	Стек	Нет	Для передачи данных из метода или в метод
Элемент массива	Куча	Есть	Элемент массива

ВИДЫ ПЕРЕМЕННЫХ

Название	Принадлежность типу или объекту	Описание (Назначение)
Локальная переменная метода	Нет	Хранит временные данные в пределах метода
Поле	Да	Хранит данные, ассоциированные с типом или его экземплярами
Параметр метода	Нет	Временная переменная, используемая для передачи данных из одного метода в другой
Элемент массива	Да	Применяется как временная переменная или как данные, ассоциированные с типом

ПРИВЕДЕНИЕ ТИПОВ

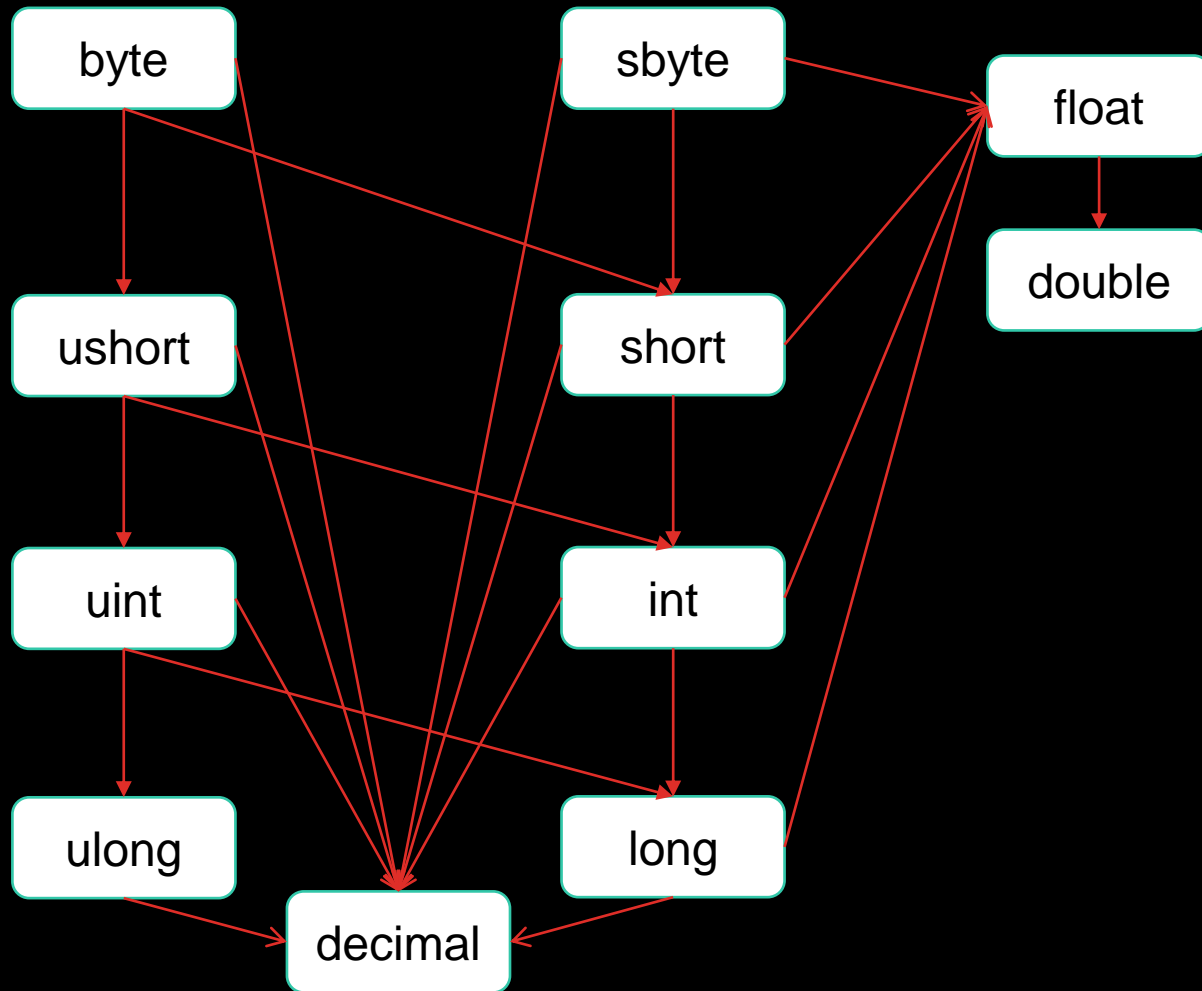
Явное и неявное приведение системных типов

Несоответствие типов

Переполнение типов



ФРАГМЕНТ ТАБЛИЦЫ НЕЯВНЫХ ПРЕОБРАЗОВАНИЙ



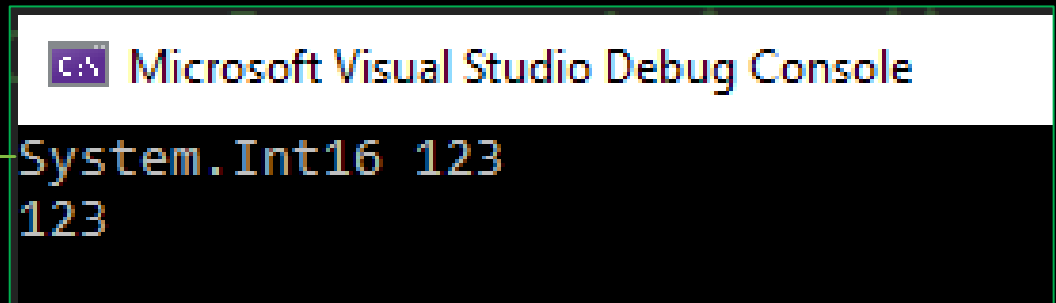
откуда	Куда можем привести неявно
sbyte	short, int, long, float, double или decimal
byte	short, ushort, int, uint, long, ulong, float, double или decimal
short	int, long, float, double или decimal
ushort	int, uint, long, ulong, float, double или decimal
int	long, float, double или decimal
uint	long, ulong, float, double или decimal
long	float, double или decimal
char	ushort, int, uint, long, ulong, float, double или decimal
float	double
ulong	float, double или decimal

НЕЯВНОЕ ПРИВЕДЕНИЕ ТИПОВ

- Обычно расширяющее, то есть значение типа с меньшим диапазоном значений приводится к типу с более широким диапазоном значений
 - Это означает, что не произойдёт потеря точности

```
short testVal = 123;  
Console.WriteLine(testVal.GetType() + " " + testVal);  
int testVal2 = testVal;  
Console.WriteLine(testVal2);
```

Обратите внимание на
название типа данных



```
Microsoft Visual Studio Debug Console  
System.Int16 123  
123
```

ЯВНОЕ ПРИВЕДЕНИЕ ТИПОВ

```
byte a = 250, b = 250;
```

```
ushort c = a + b;
```

```
Console.Write((a + b).GetType());
```

```
System.Int32
```

```
ushort c = (ushort)(a + b);
```

Общий вид операции явного приведения типов:
(<целевой_тип>)(выражение)

ОПЕРАЦИЯ ПРИВЕДЕНИЯ ТИПА

1

```
double myPi = 3.1418281828;  
int b = (int)myPi;  
Console.Write(b);
```

2

```
char ch = 'A';  
int b = ch;  
Console.Write(b);
```

3

```
int b = 98;  
char ch = b;  
Console.Write(ch);
```


КЛАСС CONVERT

- Для преобразования одного базового типа в другой в рамках платформы .NET рекомендуется использовать методы `System.Convert`

Что будет выведено?

```
double t = 11.61152;  
  
Console.WriteLine((int)t);  
Console.WriteLine(Convert.ToInt32(t));
```

FormatException

OverflowException

```
public static int ToInt32 (string? value);
```

Класс Convert [<https://learn.microsoft.com/ru-ru/dotnet/api/system.convert?view=net-7.0>]

ВАРИАНТЫ ЗАВЕРШЕНИЯ МЕТОДА CONVERT() В БАЗОВЫЕ ТИПЫ



Без
преобразования

Вернёт

Экземпляр исходного
типа



Успешное
преобразования

Вернёт

Значение целевого типа



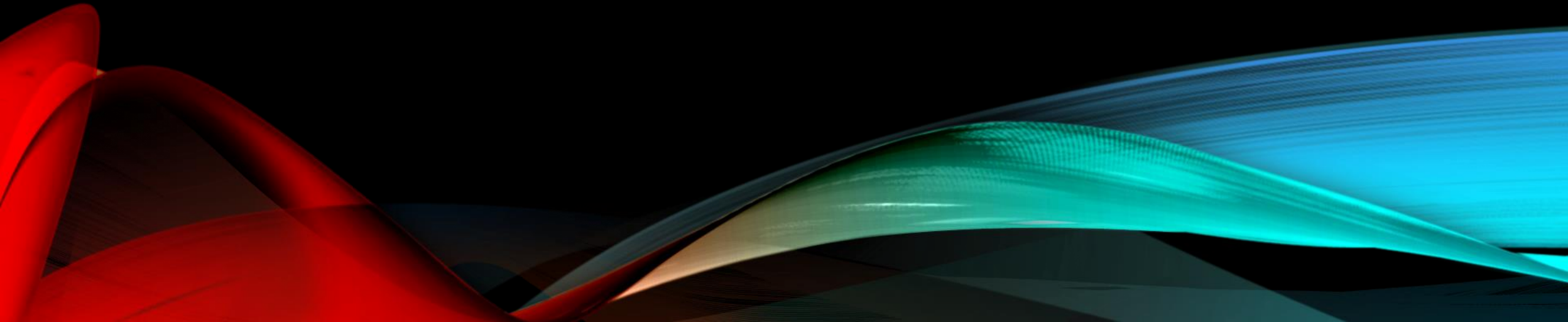
Исключение

Создаст объект

- `InvalidCastException`
- `FormatException`
- `OverflowException`

ПОЛУЧЕНИЕ ДАННЫХ ОТ ПОЛЬЗОВАТЕЛЯ

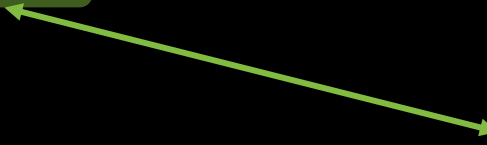
Приведение строк в простые встроенные типы



ПОЛУЧЕНИЕ ДАННЫХ ОТ ПОЛЬЗОВАТЕЛЯ

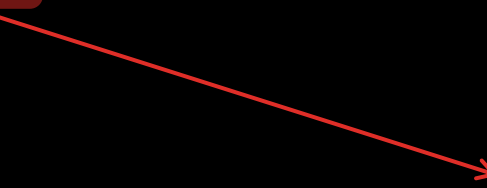
Считать или записать в консольное окно можно только строку (**string**)

string Console.ReadLine()



```
Console.WriteLine("Привет, как тебя зовут?"); // Вывели строку.  
string input = Console.ReadLine(); // Получили строку.  
// Собрали и вывели новую строку.  
Console.WriteLine($"Рад видеть тебя, {input}");
```

char Console.Read()



```
string a = Console.ReadLine();  
string b = Console.Read(); // так нельзя  
string c = Console.Read().ToString(); // а так можно
```

ПРИМЕР. ИСПОЛЬЗОВАНИЕ INT.PARSE()

<тип> <тип>.Parse(<строка>)

Мы хотим вывести на экран сумму двух чисел, попробуем сделать это без использования ЦЕЛЫХ ТИПОВ:

```
string op1, op2; // два операнда выражения
op1 = Console.ReadLine();
op2 = Console.ReadLine();
Console.WriteLine(op1 + op2);
```

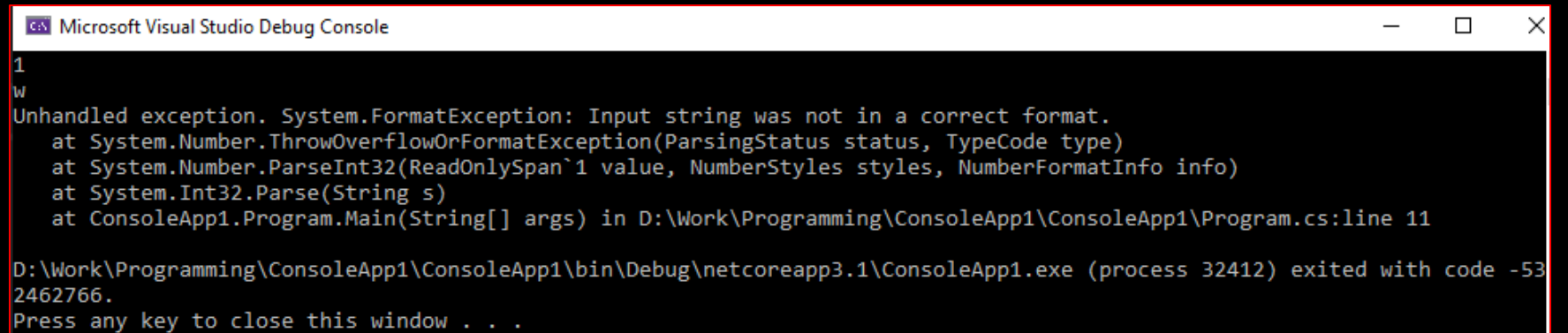
Используем метод **int.Parse()**, самостоятельно попробуйте использовать некорректное значение при вводе, например, букву:

```
int op1, op2; // два операнда выражения
op1 = int.Parse(Console.ReadLine());
op2 = int.Parse(Console.ReadLine());
Console.WriteLine(op1 + op2);
```

ИСКЛЮЧЕНИЯ ПРИ ВВОДЕ ДАННЫХ

```
int op1, op2; // два операнда выражения
op1 = int.Parse(Console.ReadLine());
op2 = int.Parse(Console.ReadLine());
Console.WriteLine(op1 + op2);
```

Если вы попробовали вместо цифр вводить другие значения, то наверняка получили сообщение об аварийном завершении программы:

A screenshot of the Microsoft Visual Studio Debug Console window. The window title is "Microsoft Visual Studio Debug Console". The text inside shows the start of a program execution with "1" and "w" on separate lines. Then, an "Unhandled exception" occurs: "System.FormatException: Input string was not in a correct format." The stack trace lists the following frames: "at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)", "at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)", "at System.Int32.Parse(String s)", and "at ConsoleApp1.Program.Main(String[] args) in D:\Work\Programming\ConsoleApp1\ConsoleApp1\Program.cs:line 11". Below the stack trace, it says "D:\Work\Programming\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 32412) exited with code -532462766." and "Press any key to close this window . . .".

```
Microsoft Visual Studio Debug Console
1
w
Unhandled exception. System.FormatException: Input string was not in a correct format.
   at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)
   at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)
   at System.Int32.Parse(String s)
   at ConsoleApp1.Program.Main(String[] args) in D:\Work\Programming\ConsoleApp1\ConsoleApp1\Program.cs:line 11

D:\Work\Programming\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 32412) exited with code -532462766.
Press any key to close this window . . .
```


ОБРАБОТКА ИСКЛЮЧЕНИЙ ВВОДА

```
int op1, op2; // два операнда выражения
try {
    op1 = int.Parse(Console.ReadLine());
    op2 = int.Parse(Console.ReadLine());
}
catch (FormatException ex) {
    Console.WriteLine("Один из операндов задан некорректно");
    return;
}
Console.WriteLine(op1 + op2);
```

Этот тип мы подглядели в исключении, но правильнее смотреть его в документации (<https://docs.microsoft.com/ru-ru/dotnet/api/system.int32.parse?view=net-5.0>)

ПРИВЕДЕНИЕ ТИПОВ

```
int firstInt, secondInt;  
Console.Write("Целое число: "); // Ввод первого числа.  
string InputStr = Console.ReadLine();  
firstInt = int.Parse(InputStr); // Преобразование в тип int.  
  
Console.Write("Целое число: "); // Ввод представления второго числа.  
InputStr = Console.ReadLine();  
int.TryParse(InputStr, out secondInt);  
  
Console.WriteLine("Ваш текст: " + firstInt + secondInt);
```

```
C:\Windows\system32\cmd.exe  
Целое число: 11  
Целое число: 22  
Ваш текст: 1122  
Для продолжения нажмите любую клавишу . . .
```

Самостоятельно проведите эксперименты с программой:

1. Ввести вместо первого числа букву
2. Ввести вместо второго числа букву
3. Заключить `firstInt + secondInt` в круглые скобки



ПРИМЕР. ИСПОЛЬЗОВАНИЕ INT.TRYPARSE()

`bool <тип>.TryParse(<строка>, out <переменная>)`

В использовании метода **TryParse()** есть тонкость. Он дружелюбно не вызывает аварийных ситуаций, но в случае, если значение не приводимо к типу, заменяет его нулём целевого типа

В коде ниже, это приведёт к неверной работе программы, попробуйте проверить это самостоятельно:

```
int op1, op2; // два операнда выражения
int.TryParse(Console.ReadLine(), out op1);
int.TryParse(Console.ReadLine(), out op2);
Console.WriteLine(op1 + op2);
```

ФОРМИРОВАНИЕ ПРЕДСТАВЛЕНИЯ НЕДЕСЯТИЧНЫХ ЧИСЕЛ

```
int bin = 2, oct = 8, hex = 16, testVal = 1024;  
Console.WriteLine(Convert.ToString(testVal, bin));  
Console.WriteLine(Convert.ToString(testVal, oct));  
Console.WriteLine(Convert.ToString(testVal, hex));
```

100000000000

2000

400

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- Обзор среды CLR (<https://docs.microsoft.com/ru-ru/dotnet/standard/clr>)
- Richter, J. CLR via C#. Fourth edition
- *Таблица целых типов (Справочник по C#)*
- [[https://docs.microsoft.com/ru-ru/previous-versions/visualstudio/visual-studio-2008/exx3b86w\(v=vs.90\)?redirectedfrom=MSDN](https://docs.microsoft.com/ru-ru/previous-versions/visualstudio/visual-studio-2008/exx3b86w(v=vs.90)?redirectedfrom=MSDN)]
- Scope of Variable (<https://www.geeksforgeeks.org/scope-of-variables-in-c-sharp/>)