

Введение в программирование на языке C#

Основные понятия языка C#

Состав языка:

Символы

Лексемы: константы, имена, ключевые слова, разделители

Выражения

Операторы

Состав языка

■ Символы:

- буквы: A-Z, a-z, _, бук
- цифры: 0-9, A-F
- спец. символы: +, *, {, ...
- пробельные символы

Лексема (token, токен) –
минимальная единица языка,
имеющая самостоятельный
смысл

■ Лексемы:

- константы
- имена Vasia a "Вася" _11
- ключевые слова double do if
- знаки операций + <= new
- разделители ; [] ,

■ Выражения

- выражение - правило вычисления значения: a + b

■ Операторы

- исполняемые: c = a + b;
- описания: double a, b;

Константы (литералы) C#

Вид	Примеры
<u>Булевские</u>	<u>true</u> <u>false</u>
<u>Целые</u> десятич.	8 199226 0 Lu
<u>16-ричн.</u>	<u>0xA</u> <u>0x1B8</u> <u>0X00FFL</u>
<u>Веществ.</u> с тчк	5.7 .001f 35m
<u>с порядком</u>	<u>0.2E6</u> .11e-3 <u>5E12</u>
<u>Символьные</u>	<u>'A'</u> <u>'\x74'</u> <u>'\0'</u> <u>'\\'</u> <u>'\uA81B'</u>
<u>Строковые</u>	"Здесь был Vasia"
	"\tЗначение r=\xF5\n"
	был \u0056\u0061"
	@ "C:\temp\file1.txt"
<u>Константа null</u>	null

Суффикс типа

5·10¹²

Управляющий символ

Кодировка Unicode

Имена (идентификаторы)

- имя должно начинаться с буквы или _;
- имя должно содержать только буквы, знак подчеркивания и цифры;
- прописные и строчные буквы различаются;
- длина имени практически не ограничена.
- имена не должны совпадать с ключевыми словами, однако допускается: @if, @float...
- в именах можно использовать управляющие последовательности Unicode

Примеры правильных имен:

Vasia, Вася, _13, \u00F2\u01DD, @while.

Примеры неправильных имен:

2late, Big gig, Б#г

Нотации

Понятные и согласованные между собой имена — основа хорошего стиля. Существует несколько *нотаций* — соглашений о правилах создания имен.

В C# для именованя различных видов программных объектов чаще всего используются две нотации:

- *Нотация Паскаля* - каждое слово начинается с прописной буквы:
 - `MaxLength, MyFuzzyShooshpanchik`
- *Camel notation* - с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого:
 - `maxLength, myFuzzyShooshpanchik`

Ключевые слова, знаки операций, разделители

- *Ключевые слова* — идентификаторы, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены.
 - Например, для оператора перехода определено слово `goto`.
- *Знак операции* — один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.
 - Например: **+** **-** **=** **==** **%=** **new** **is**
- Операции делятся на *унарные* (с одним операндом), *бинарные* (с двумя) и *тернарную* (с тремя).
- *Разделители* используются для разделения или, наоборот, группирования элементов. Примеры разделителей: скобки, точка, запятая.

Ключевые слова C#

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

Типы данных:

Концепция

Классификация

Встроенные типы данных

Концепция типа данных

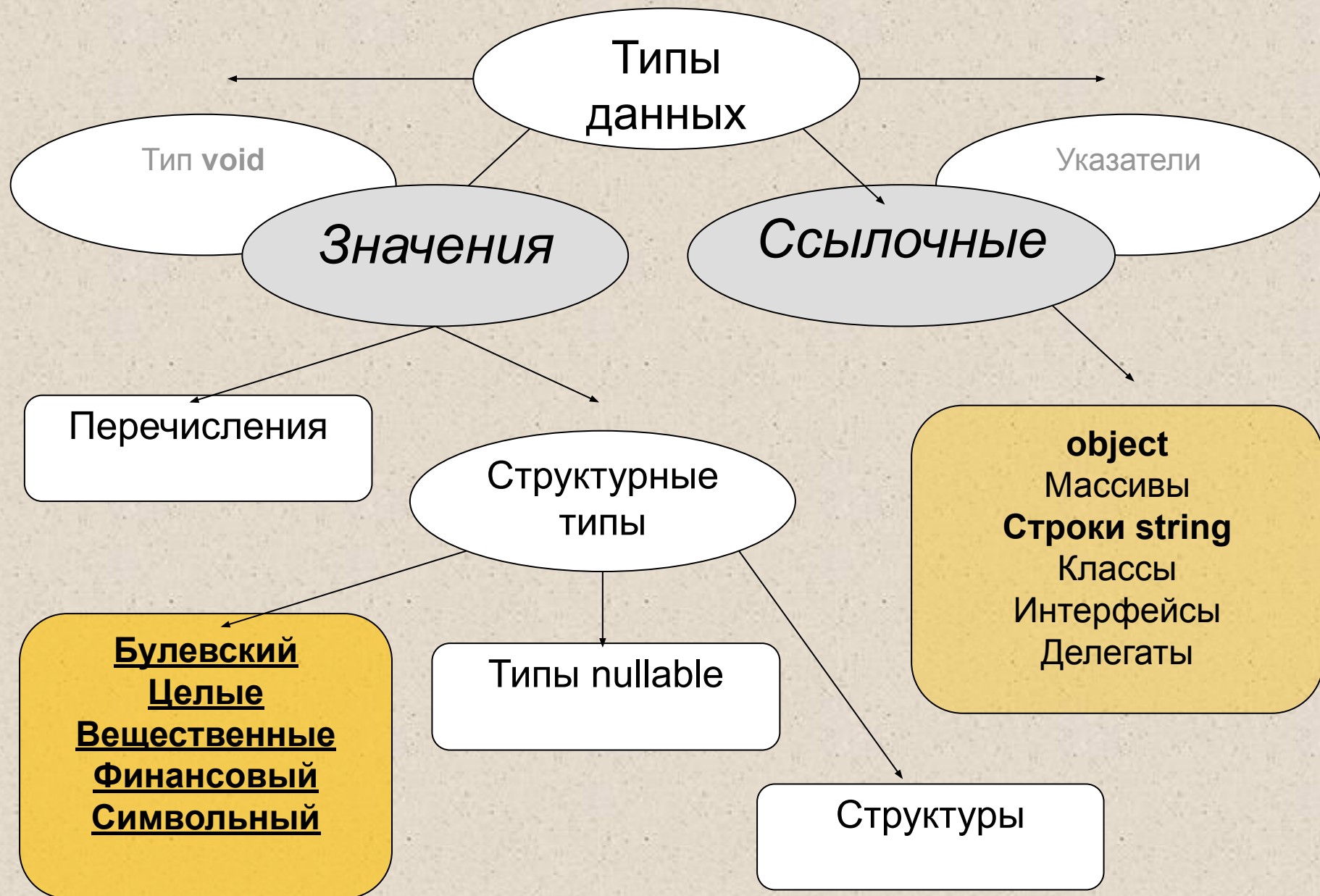
Тип данных определяет:

- **внутреннее представление данных** =>
множество их возможных значений
- **допустимые действия над данными** =>
операции и функции

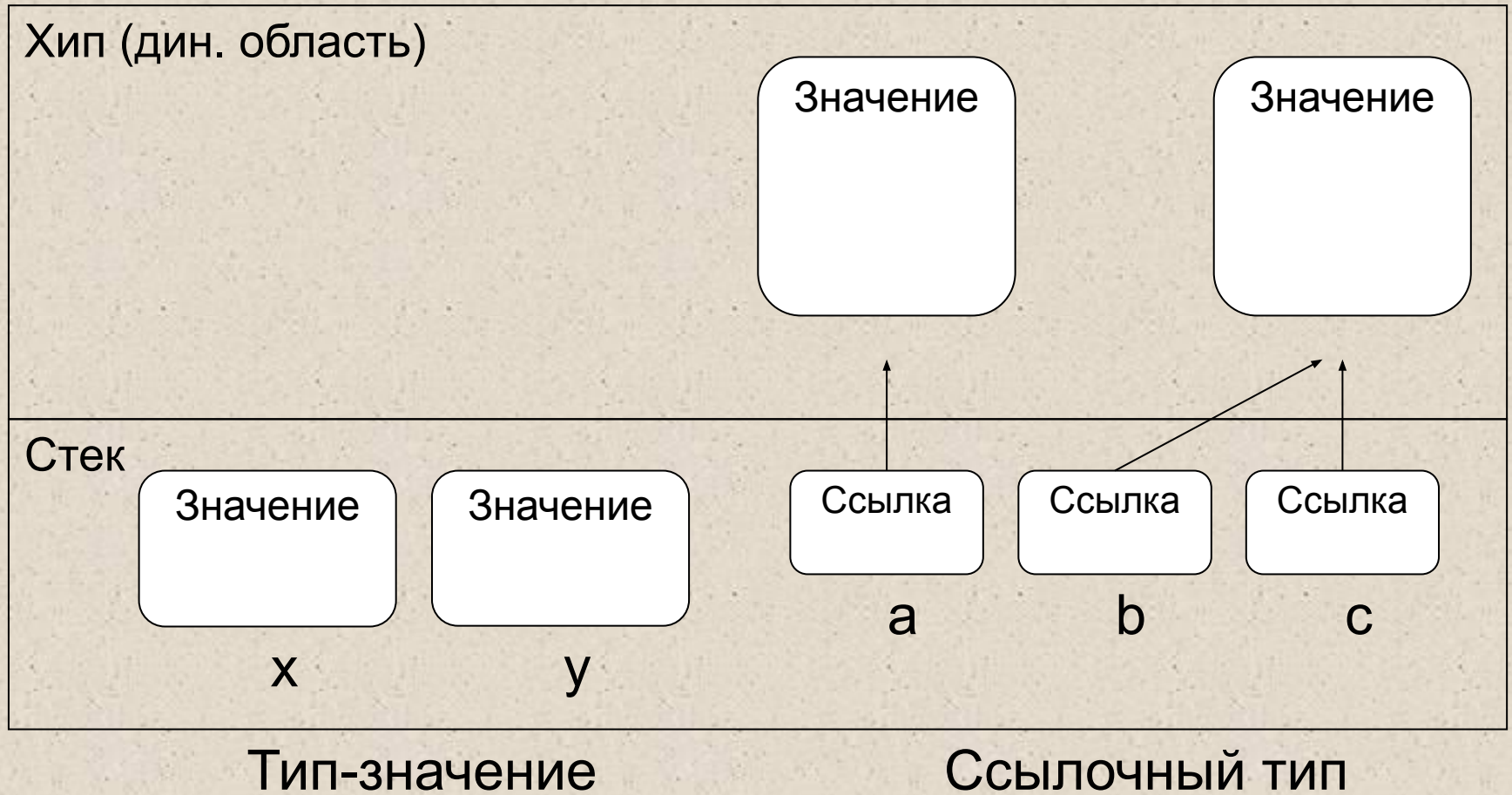
Различные классификации типов данных



Основная классификация типов C#



Хранение в памяти величин значимого и ссылочного типа



Встроенные типы данных C#:

- Булевский
- Целые
- Вещественные
- Финансовый
- Символьный
- object

Логический (булевский) и целые

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер в битах
Булевский	bool	Boolean	true, false		
Целые	sbyte	SByte	-128 — 127	знаковое	8
	byte	Byte	0 — 255	беззнаковое	8
	short	Int16	-32768 — 32767	знаковое	16
	ushort	UInt16	0 — 65535	беззнаковое	16
	int	Int32	$\approx(-2 \cdot 10^9 — 2 \cdot 10^9)$	знаковое	32
	uint	UInt32	$\approx(0 — 4 \cdot 10^9)$	беззнаковое	32
	long	Int64	$\approx(-9 \cdot 10^{18} — 9 \cdot 10^{18})$	знаковое	64
	ulong	UInt64	$\approx(0 — 18 \cdot 10^{18})$	беззнаковое	64

Остальные

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер в битах
Символьный	char	Char	U+0000 — U+ffff	символ Unicode	16
Вещественные	float	Single	$(+-)1.5 \cdot 10^{-45} — 3.4 \cdot 10^{38}$	7 цифр	32
	double	Double	$(+-) 5.0 \cdot 10^{-324} — 1.7 \cdot 10^{308}$	15-16 цифр	64
Финансовый	decimal	Decimal	$(+-) 1.0 \cdot 10^{-28} — 7.9 \cdot 10^{28}$	28-29 цифр	128
Строковый	string	String	длина ограничена объемом доступной памяти	строка из символов Unicode	
object	object	Object	можно хранить все, что угодно	всеобщий предок	

Поля и методы встроенных типов

- Любой встроенный тип C# построен на основе стандартного класса библиотеки .NET. Это значит, что у встроенных типов данных C# есть *методы и поля*. С помощью них можно, например, получить:
 - **double.MaxValue** (или `System.Double.MaxValue`) — максимальное число типа `double`;
 - **uint.MinValue** (или `System.UInt32.MinValue`) — минимальное число типа `uint`.
- В вещественных классах есть элементы:
 - положительная бесконечность **PositiveInfinity**;
 - отрицательная бесконечность **NegativeInfinity**;
 - «не является числом»: **NaN**.

Математические функции: класс Math

Имя	Описание	Результат	Пояснения
Abs	Модуль	перегружен	$ x $ записывается <code>Math.Abs(x)</code>
Acos	Арккосинус	double	<code>Acos(double x)</code>
Asin	Арсинус	double	<code>Asin(double x)</code>
Atan	Арктангенс	double	<code>Atan(double x)</code>
Atan2	Арктангенс	double	<code>Atan2(double x, double y)</code> — угол, тангенс которого есть результат деления y на x
BigMul	Произведение	long	<code>BigMul(int x, int y)</code>
Ceiling	Округление до большего целого	double	<code>Ceiling(double x)</code>
Cos	Косинус	double	<code>Cos(double x)</code>
Cosh	Гиперболический косинус	double	<code>Cosh(double x)</code>
DivRem	Деление и остаток	перегружен	<code>DivRem(x, y, rem)</code>
E	База натурального логарифма (число e)	double	2,71828182845905
Exp	Экспонента	double	e^x записывается <code>Math.Exp(x)</code>

Floor	Округление до меньшего целого	double	Floor(double x)
IEEERemainder	Остаток от деления	double	IEEERemainder(double x, double y)
Log	Натуральный логарифм	double	$\log_e x$ записывается Math.Log(x)
Log10	Десятичный логарифм	double	$\log_{10} x$ записывается Math.Log10(x)
Max	Максимум из двух чисел	перегружен	Max(x, y)
Min	Минимум из двух чисел	перегружен	Min(x, y)
PI	Значение числа π	double	3,14159265358979
Pow	Возведение в степень	double	x^y записывается Math.Pow(x, y)
Round	Округление	перегружен	Round(3.1) даст в результате 3 Round(3.8) даст в результате 4
Sign	Знак числа	int	аргументы перегружены
Sin	Синус	double	Sin(double x)
Sinh	гиперболический синус	double	Sinh(double x)
Sqrt	Квадратный корень	double	\sqrt{x} записывается Math.Sqrt(x)
Tan	Тангенс	double	Tan(double x)
Tanh	Гиперболический тангенс	double	Tanh(double x)

Линейные программы:

- Структура программы
- Переменные
- Операции
- Выражения
- Введение в обработку исключительных ситуаций
- Простейший ввод-вывод

Структура простейшей программы на C#

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            // описания и операторы, например:
            Console.Write("Превед медвед");
        }

        // описания
    }
}
```

Переменные

- *Переменная* — это величина, которая во время работы программы может изменять свое значение.
- Все переменные, используемые в программе, должны быть описаны.
- Для каждой переменной задается ее **ИМЯ** и **ТИП**:

```
int    number;  
float  x, y;  
char   option;
```

Тип переменной выбирается исходя из диапазона и требуемой точности представления данных.

Общая структура программы на C#

пространство имен

Класс А

Переменные класса

Методы класса:

Локальные переменные

...

Класс В

Переменные класса

Методы класса:

Метод **Main**

Область действия и время жизни переменных

- Переменные описываются внутри какого-л. блока:

- 1) класса

- 2) метода или блока внутри метода

Блок — код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.

- Переменные, описанные *непосредственно внутри класса*, называются **полями класса**.
- Переменные, описанные *внутри метода класса*, называются **локальными переменными**.
- **Область действия переменной** - область программы, где можно использовать переменную.
- Область действия переменной начинается в точке ее описания и длится до конца блока, внутри которого она описана.
- **Время жизни**: переменные создаются при входе в их область действия (блок) и уничтожаются при выходе.

Инициализация переменных

- При объявлении можно присвоить переменной начальное значение (*инициализировать*).

```
int number = 100;  
float x = 0.02;  
char option = 'ю';
```

При инициализации можно использовать не только константы, но и выражения — главное, чтобы на момент описания они были вычислимыми, например:

```
int b = 1, a = 100;  
int x = b * a + 25;
```

- **Поля класса** инициализируются «значением по умолчанию» (0 соответствующего типа).
- **Локальные переменные** автоматически НЕ инициализируются. Рекомендуется всегда явным образом инициализировать переменные при описании.

Пример описания переменных

```
using System;  
namespace CA1  
{  
    class Class1  
    {  
        static void Main()  
        {  
            int    i = 3;  
            double y = 4.12;  
            decimal d = 600m;  
            string s = "Вася";  
  
        }  
    }  
}
```

Именованные константы

Вместо значений констант можно (и нужно!) использовать в программе их имена.

Это облегчает читабельность программы и внесение в нее изменений:

```
const float weight = 61.5;  
const int   n       = 10;  
const float g       = 9.8;
```

Никаких чисел, кроме 0 и 1, в программе быть не должно
(нигде, кроме описаний именованных констант)

Выражения

- *Выражение* — правило вычисления значения.
- В выражении участвуют *операнды*, объединенные знаками операций.
- Операндами выражения могут быть константы, переменные и вызовы функций.
- Операции выполняются в соответствии с *приоритетами*.
- Для изменения порядка выполнения операций используются *круглые скобки*.
- Результатом выражения всегда является значение определенного типа, который определяется типами операндов.
- Величины, участвующие в выражении, должны быть *совместимых типов*.

■ `t + Math.Sin(x)/2 * x`

результат имеет
вещественный тип

■ `a <= b + 2`

результат имеет
логический тип

■ `x > 0 && y < 0`

результат имеет
логический тип

Ассоциативность выражений

- Слева направо
- $a + b - c + d$
- $(((a + b) - c) + d)$
- $a * b / c * d$
- $(((a * b) / c) * d)$

- Справа налево
- $a = b = c = d$
- $(a = (b = (c = d)))$

$$\frac{a * b}{c} * d$$

Приоритеты операций C#

1. Первичные `()`, `[]`, `++`, `--`, `new`, ...
2. Унарные `~`, `!`, `++`, `--`, `-`, ...
3. Типа умножения (мультипликативные) `*`, `/`, `%`
4. Типа сложения (аддитивные) `+`, `-`
5. Сдвига `<<`, `>>`
6. Отношения и проверки типа `<`, `>`, `is`, ...
7. Проверки на равенство `==`, `!=`
8. Поразрядные логические `&`, `^`, `|`
9. Условные логические `&&`, `||`
10. Условная `?:`
11. Присваивания `=`, `*=`, `/=`, ...

Тип результата выражения

- Если входящие в выражение **операнды одного типа** и операция для этого типа определена, то результат выражения будет иметь тот же тип.
- Если **операнды разного типа** и (или) операция для этого типа не определена, перед вычислениями автоматически выполняется **преобразование типа** по правилам, обеспечивающим приведение *более коротких типов к более длинным* для сохранения значимости и точности.

```
char  c = 'A';
```

```
int   i = 100;
```

```
double d = 1;
```

```
double summa = c + i + d; // 166
```

Явное преобразование типа

- Автоматическое (*неявное*) преобразование возможно не всегда, а только если при этом не может случиться потеря значимости.
- Если неявного преобразования из одного типа в другой не существует, программист может задать *явное* преобразование типа с помощью операции **(тип) x**.

```
char  c = 'A';
```

```
int    i = 100;
```

```
double d = 1;
```

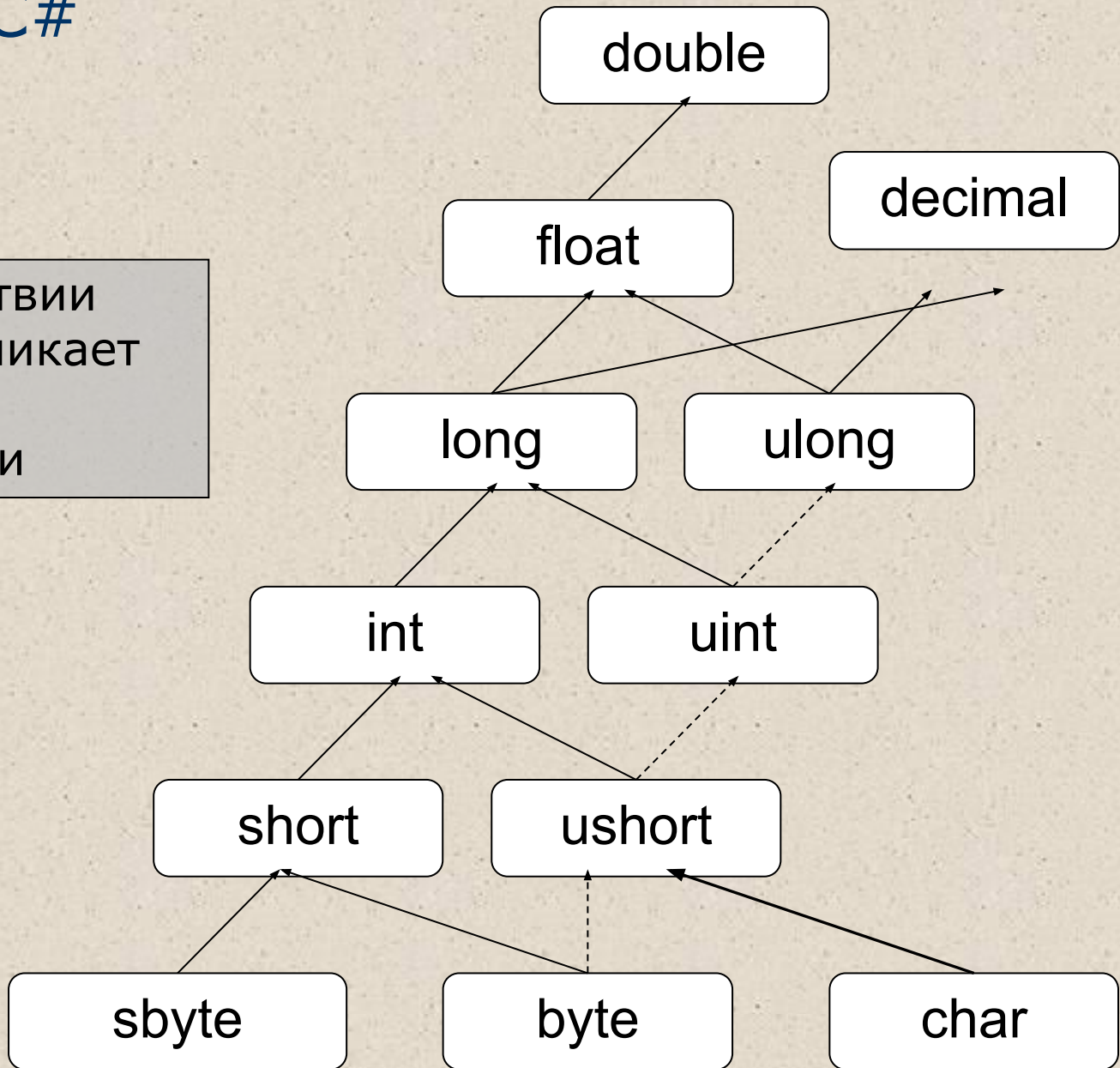
```
c = (char) i;           // 'd'
```

```
c = (char) d;
```

```
i = (int) d;
```


Неявные арифметические преобразования типов в C#

при отсутствии
линии возникает
ошибка
компиляции



Понятие «исключительная ситуация»

- При вычислении выражений могут возникнуть ошибки (переполнение, деление на ноль).
- В C# есть механизм **обработки исключительных ситуаций (исключений)**, который позволяет избегать аварийного завершения программы.
- Если в процессе вычислений возникла ошибка, система сигнализирует об этом с помощью **выбрасывания (генерирования) исключения**.
- Каждому типу ошибки соответствует свое исключение. Исключения являются классами, которые имеют общего предка — класс **Exception**, определенный в пространстве имен System.
- Например, при делении на ноль будет выброшено исключение `DivideByZeroException`, при переполнении — исключение `OverflowException`.
- В программе необходимо предусмотреть **обработку исключений**.

Инкремент и декремент

```
using System;
namespace CA1
{
    class C1
    {
        static void Main()
        {
            int x = 3, y = 3;
            Console.Write( "Значение префиксного выражения: " );
            Console.WriteLine( ++x );
            Console.Write( "Значение x после приращения: " );
            Console.WriteLine( x );

            Console.Write( "Значение постфиксного выражения: " );
            Console.WriteLine( y++ );
            Console.Write( "Значение y после приращения: " );
            Console.WriteLine( y );
        }
    }
}
```

Результат работы программы:
Значение префиксного выражения: 4
Значение x после приращения: 4
Значение постфиксного выражения: 3
Значение y после приращения: 4

Операция new

Служит для **создания** нового объекта (*выделение памяти в хипе + инициализация*).

Формат:

new тип ([аргументы])

С помощью этой операции можно создавать объекты как ссылочных, так и значимых типов, например:

```
object z = new object();
```

```
Monster Вася = new Monster(100);
```

```
int i = new int();           // то же самое, что int i = 0;
```

Для значимых типов обычно не применяется.

Операции отрицания

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            sbyte a = 3, b = -63, c = 126;
            bool d = true;
            Console.WriteLine( -a ); // Результат -3
            Console.WriteLine( -c ); // Результат -126
            Console.WriteLine( !d ); // Результат false
            Console.WriteLine( ~a ); // Результат -4
            Console.WriteLine( ~b ); // Результат 62
            Console.WriteLine( ~c ); // Результат -127
        }
    }
}
```

a = 00000011 - ! ~
~a = 11111100

b = 11000001
~b = 00111110

c = 01111110
~c = 10000001

Явное преобразование типа

- `long b = 300;`
- `int a = (int) b;` // данные не теряются
- `byte d = (byte) a;` // данные теряются

Умножение *

- Возвращает результат перемножения двух операндов.
- Стандартная операция умножения определена для типов `int`, `uint`, `long`, `ulong`, `float`, `double` и `decimal`.
- К величинам других типов ее можно применять, если для них возможно неявное преобразование к этим типам.
- Тип результата операции равен «наибольшему» из типов операндов, но не менее `int`.
- Если оба операнда *целочисленные* или типа `decimal` и результат операции слишком велик для представления с помощью заданного типа, генерируется исключение `System.OverflowException`

Результаты вещественного умножения

*	$+y$	$-y$	$+0$	-0	$+\infty$	$-\infty$	NaN
$+x$	$+z$	$-z$	$+0$	-0	$+\infty$	$-\infty$	NaN
$-x$	$-z$	$+z$	-0	$+0$	$-\infty$	$+\infty$	NaN
$+0$	$+0$	-0	$+0$	-0	NaN	NaN	NaN
-0	-0	$+0$	-0	$+0$	NaN	NaN	NaN
$+\infty$	$+\infty$	$-\infty$	NaN	NaN	$+\infty$	$-\infty$	NaN
$-\infty$	$-\infty$	$+\infty$	NaN	NaN	$-\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Пример (умножение *, деление /, остаток %)

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            int x = 11, y = 4;
            float z = 4;
            Console.WriteLine( z * y );           // Результат 16
            Console.WriteLine( z * 1e308 );      // Рез. "бесконечность"
            Console.WriteLine( x / y );         // Результат 2
            Console.WriteLine( x / z );         // Результат 2,75
            Console.WriteLine( x % y );          // Результат 3
            Console.WriteLine( 1e-324 / 1e-324 ); // Результат NaN
        }
    }
}
```

Операции сдвига (<< и >>)

- Применяются к целочисленным операндам.
- Сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом.
- При *сдвиге влево* (<<) освободившиеся разряды обнуляются.
- При *сдвиге вправо* (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае.
- Стандартные операции сдвига определены для типов `int`, `uint`, `long` и `ulong`.

Пример

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            byte a = 3, b = 9;
            sbyte c = 9, d = -9;
            Console.WriteLine( a << 1 );           // Результат 6
            Console.WriteLine( a << 2 );           // Результат 12
            Console.WriteLine( b >> 1 );           // Результат 4
            Console.WriteLine( c >> 1 );           // Результат 4
            Console.WriteLine( d >> 1 );           // Результат -5
        }
    }
}
```

00001100

Операции отношения и проверки на равенство

- *Операции отношения* ($<$, \leq , $>$, \geq , $==$, $!=$) сравнивают первый операнд со вторым.
- Операнды должны быть арифметического типа.
- Результат операции — логического типа, равен true или false.

$x == y$ -- true, если x равно y , иначе false

$x != y$ -- true, если x не равно y , иначе false

$x < y$ -- true, если x меньше y , иначе false

$x > y$ -- true, если x больше y , иначе false

$x \leq y$ -- true, если x меньше или равно y , иначе false

$x \geq y$ -- true, если x больше или равно y , иначе false

Условные логические операции && и ||

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            Console.WriteLine( true && true );    // Результат true
            Console.WriteLine( true && false );    // Результат false
            Console.WriteLine( true || true );    // Результат true
            Console.WriteLine( true || false );    // Результат true
        }
    }
}
```

$(a < b) \ || \ (a < c)$

Если первое условие истинно,
второе не вычисляется

Условная операция

■ **операнд_1 ? операнд_2 : операнд_3**

Операнд_1 — выражение, для которого существует неявное преобразование к логическому типу.

Если результат вычисления операнда_1 равен true, результат операции — значение **операнда_2**,
иначе — значение **операнда_3**.

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            int a = 11, b = 4;
            int max = b > a ? b : a;
            Console.WriteLine( max );    // Результат 11
        }
    }
}
```

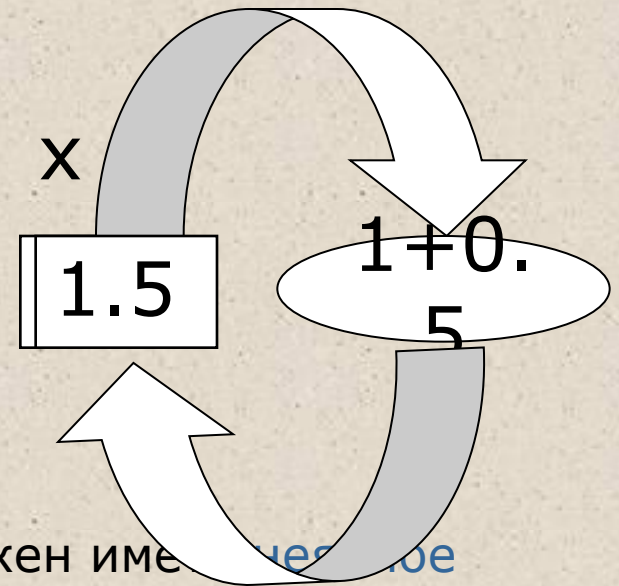
Операция присваивания

Присваивание – это замена старого значения переменной на новое. Старое значение стирается бесследно.

Операция может использоваться в программе как законченный оператор.

переменная = выражение

```
a = b + c;  
x = 1;  
x = x + 0.5;
```



Правый операнд операции присваивания должен иметь **числовое преобразование** к типу левого операнда, например:

вещественная переменная = целое выражение;

Сложное присваивание в C#

- $x += 0.5;$ соответствует $x = x + 0.5;$
- $x *= 0.5;$ соответствует $x = x * 0.5;$
- $a \% = 3;$ соответствует $a = a \% 3;$
- $a << = 2;$ соответствует $a = a << 2;$

и т.п.

Консольный ввод-вывод

Приложение: консольное или Windows?

```
class A
```

```
{
```

```
    public static void Main()
```

```
    {
```

```
        System.Windows.Forms.MessageBox.Show("Hello!"  
);
```

```
    }
```

```
}
```

```
// Необходимо добавить ссылку на  
    System.Windows.Forms
```

```
// (Add Reference...)
```

Вывод на консоль – 1/4

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            int    i = 3;
            double y = 4.12;
            decimal d = 600m;
            string  s = "Вася";
```

```
        Console.Write( i );
        Console.WriteLine( " y = " + y);
        Console.WriteLine("d = " + d + " s = " + s );
```

```
    }
```

```
}
```

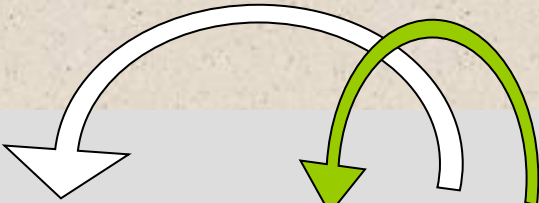
Результат работы программы:
3 y = 4,12
d = 600 s = Вася

Вывод на консоль – 2/4

```
using System;  
namespace A  
{  
    class Class1  
    {  
        static void Main()  
        {  
            int i = 3;  
            double y = 4.12;  
            decimal d = 600m;  
            string s = "Вася";
```

Результат работы программы:
3 y = 4,12
d = 600 s = Вася

```
Console.Write( i );  
Console.Write( " y = {0} \nd = {1}", y, d );  
Console.WriteLine( " s = " + s );
```



```
}
```

```
}
```

```
}
```

Вывод на консоль – 3/4

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            int i = 3;
            double y = 4.12;
            decimal d = 600m;
            string s = "Вася";
```

Результат работы программы:
3 y = 4,12
d = 600 s = Вася

```
Console.Write( i );
Console.Write( " y = {0:F2} \nd = {1:D3}", y, d );
Console.WriteLine( " s = " + s );
```

Формат

Формат

```
    }
}
}
```

Вывод на консоль – 4/4

```
using System;
```

```
namespace A
```

```
{ class Class1
```

```
{ static void Main()
```

```
{
```

```
int i = 3;
```

```
double y = 4.12;
```

```
decimal d = 600m;
```

```
string s = "Вася";
```

```
Console.Write( " y = {0,5:0.# ' руб. ' } \n", y );
```

```
}
```

```
}
```

```
}
```

y = 4,1 руб.

пользовательский
формат

5 позиций под
значение

Формат местозаполнителя

{ номер [, длина] [: формат] }

- **номер** – номер элемента в списке вывода (может идти не по порядку и повторяться)
- **длина** – количество позиций под значение. Если длина отрицательная, значение выравнивается по левому краю, иначе - по правому.
- **формат** – строка формата для выводимого значения (описатели формата на следующем слайде)
- Примеры:
 - {0,-6:F4}
 - {2:X4}
 - {1:hh}

Описатели формата – 1/3

	Имя	Описание	Примеры
"C" или "c"	Валюта	Результат: значение валюты. Поддерживается: всеми числовыми типами данных. Описатель точности: количество цифр дробной части.	123.456 ("C", en-US) -> \$123.46 123.456 ("C", fr-FR) -> 123,46 € 123.456 ("C", ja-JP) -> ¥123 -123.456 ("C3", en-US) -> (\$123.456) -123.456 ("C3", fr-FR) -> -123,456 € -123.456 ("C3", ja-JP) -> -¥123.456
"D" или "d"	Десятичное число	Результат: целочисленные цифры с необязательным отрицательным знаком. Поддерживается: только целочисленными типами данных. Описатель точности: минимальное число цифр. Описатель точности по умолчанию: минимальное необходимое число цифр.	1234 ("D") -> 1234 -1234 ("D6") -> -001234
"E" или "e"	Экспоненциальный (научный)	Результат: экспоненциальная нотация. Поддерживается: всеми числовыми типами данных. Описатель точности: количество цифр дробной части. Описатель точности по умолчанию: 6.	1052.0329112756 ("E", en-US) -> 1.052033E+003 1052.0329112756 ("e", fr-FR) -> 1,052033e+003 -1052.0329112756 ("e2", en-US) -> -1.05e+003 -1052.0329112756 ("E2", fr-FR) -> -1,05E+003

Описатели формата – 2/3

<p>"F" или "f"</p>	<p>Фиксированная запятая</p>	<p>Результат: цифры целой и дробной частей с необязательным отрицательным знаком. Поддерживается: всеми числовыми типами данных. Описатель точности: количество цифр дробной части.</p>	<p>1234.567 ("F", en-US) -> 1234.57 1234.567 ("F", de-DE) -> 1234,57 1234 ("F1", en-US) -> 1234.0 1234 ("F1", de-DE) -> 1234,0 -1234.56 ("F4", en-US) -> -1234.5600 -1234.56 ("F4", de-DE) -> -1234,5600</p>
<p>"G" или "g"</p>	<p>Общие</p>	<p>Результат: наиболее компактная запись из двух вариантов: экспоненциального и с фиксированной запятой. Поддерживается: всеми числовыми типами данных. Описатель точности: количество значащих цифр. Описатель точности по умолчанию: определяется численным типом.</p>	<p>-123.456 ("G", en-US) -> -123.456 123.456 ("G", sv-SE) -> -123,456 123.4546 ("G4", en-US) -> 123.5 123.4546 ("G4", sv-SE) -> 123,5 -1.234567890e-25 ("G", en-US) -> -1.23456789E-25 -1.234567890e-25 ("G", sv-SE) -> -1,23456789E-25</p>
<p>"N" или "n"</p>	<p>Номер</p>	<p>Результат: цифры целой и дробной частей, разделители групп и разделитель целой и дробной частей с необязательным отрицательным знаком. Поддерживается: всеми числовыми типами данных. Описатель точности: желаемое число знаков дробной части.</p>	<p>1234.567 ("N", en-US) -> 1,234.57 1234.567 ("N", ru-RU) -> 1 234,57 1234 ("N", en-US) -> 1,234.0 1234 ("N", ru-RU) -> 1 234,0 -1234.56 ("N", en-US) -> -1,234.560 -1234.56 ("N", ru-RU) -> -1 234,560</p>

Описатели формата – 3/3

<p>"P" или "p"</p>	<p>Процент</p>	<p>Результат: число, умноженное на 100 и отображаемое с символом процента. Поддерживается: всеми числовыми типами данных. Описатель точности: желаемое число знаков дробной части.</p>	<p>1 ("P", en-US) -> 100.00 % 1 ("P", fr-FR) -> 100,00 % -0.39678 ("P1", en-US) -> -39.7 % -0.39678 ("P1", fr-FR) -> -39,7 %</p>
<p>"R" или "r"</p>	<p>Приемо-передача</p>	<p>Результат: строка, дающая при обратном преобразовании идентичное число. Поддерживается: Single, Double и BigInteger. Описатель точности: игнорируется.</p>	<p>123456789.12345678 ("R") -> 123456789.12345678 -1234567890.12345678 ("R") -> -1234567890.1234567</p>
<p>"X" или "x"</p>	<p>Шестнадцатеричный</p>	<p>Результат: шестнадцатеричная строка. Поддерживается: только целочисленными типами данных. Описатель точности: число цифр в результирующей строке.</p>	<p>255 ("X") -> FF -1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF</p>
<p>Любой другой символ</p>	<p>Неизвестный описатель</p>	<p>Результат: порождение исключения FormatException во время выполнения.</p>	

Ввод с консоли – 1/2

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            string s = Console.ReadLine();           // ввод строки

            char c = (char)Console.Read();           // ввод символа
            Console.ReadLine();

            string buf;                               // буфер для ввода чисел
            buf = Console.ReadLine();

            int i = Convert.ToInt32( buf );         // преобразование в целое

            buf = Console.ReadLine();
            double x = Convert.ToDouble( buf ); // преобразование в вещ.

            buf = Console.ReadLine();
            double y = double.Parse( buf );       // преобразование в вещ.
        }
    }
}
```

Ввод с консоли – 2/2

```
using System;
namespace A
{
    class Class1
    {
        static void Main()
        {
            string s = Console.ReadLine();           // ввод строки

            char c = (char)Console.Read();           // ввод символа
            Console.ReadLine();

            int i = Convert.ToInt32( Console.ReadLine() );

            double x = Convert.ToDouble( Console.ReadLine() );

            double y = double.Parse( Console.ReadLine() );

        }
    }
}
```

Пример: перевод температуры из F в C

```
using System;
```

```
namespace CA1
```

```
{ class Class1
```

```
{ static void Main()
```

```
{
```

```
    Console.WriteLine( "Введите температуру по Фаренгейту" );
```

```
    double fahr = Convert.ToDouble( Console.ReadLine() );
```

```
    double cels = 5.0 / 9 * (fahr - 32);
```

```
    Console.WriteLine( "По Фаренгейту: {0} в градусах Цельсия: {1}",  
        fahr, cels );
```

```
}
```

```
}
```

```
}
```

$$C = \frac{5}{9}(F - 32)$$

Домашнее задание

Изучить темы:

- Виды констант в C#
- Понятие «тип данных»; встроенные типы C#
- Переменные: описание, инициализация, область действия, время жизни
- Операции: приоритеты, тип результата
- Неявные преобразования типов
- Простейший ввод и вывод (класс Console)

Примечание: «изучить» - прочитать в учебнике, найти в стандарте, найти в справке, понять, применить в программе, уложить в голове в систему, объяснить бабушке.