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 “Вася”
  - имена Vasia a _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
Булевские	**true** **false**
Целые десятич.	**8** 199226
16-ричн.	0xA 0x1B8
Веществ. с тчк	**5.7**
с порядком	**0.2E6** .11e–3 5E12
Символьные	**'A'** '**\x**74' '\0' '\\'
Строковые	**"Здесь был Vasia"**
	"Здесь был \u0056\u0061"
	@"C:\temp\file1.txt"
Константа null	null

$5 \cdot 10^{12}$

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 именования 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 return sbyte sealed short sizeof
stackalloc static string struct switch this throw
true try typeof uint ulong unchecked unsafe
ushort using virtual void volatile while

Типы данных:

Концепция

Классификация

Встроенные типы данных

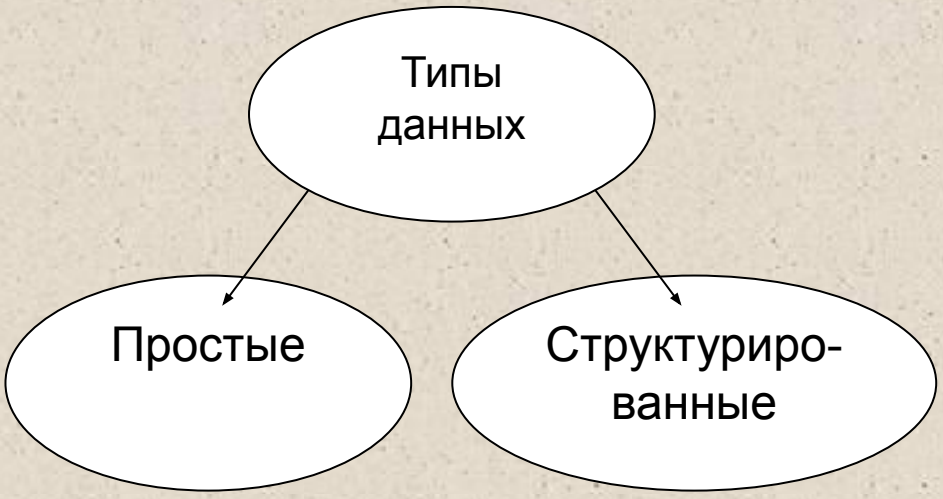
Концепция типа данных

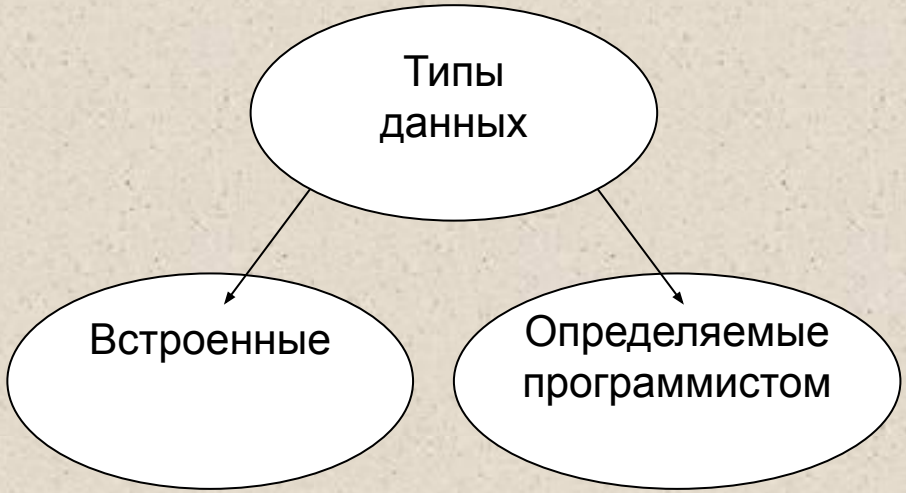
Тип данных определяет:

- **внутреннее представление данных** =>
  множество их возможных значений

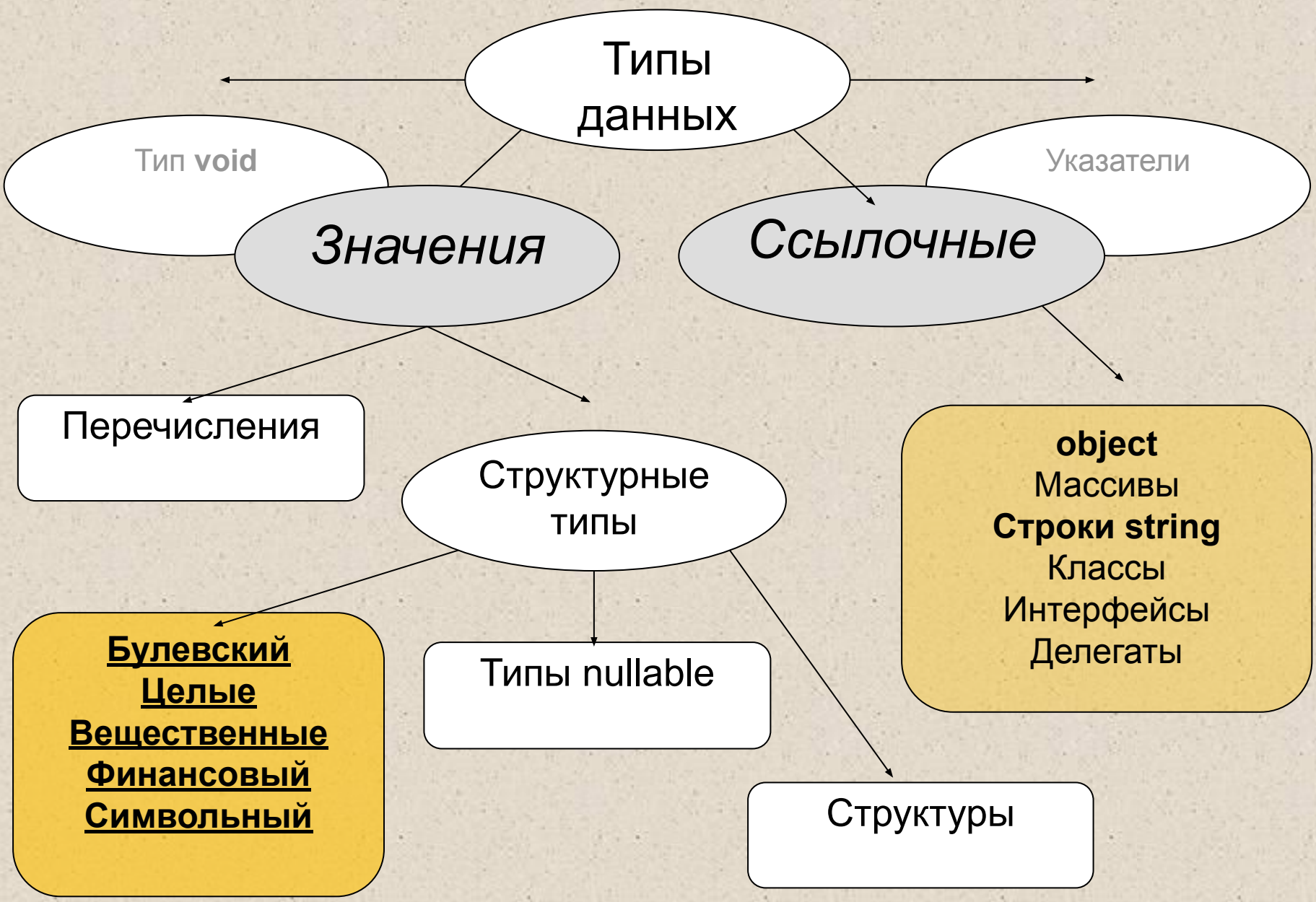
- **допустимые действия над данными** =>
  операции и функции

Различные классификации типов данных





Основная классификация типов C#



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
Веществен-ные	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	

Математические функции: класс Math

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

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#

Область действия и время жизни переменных

- Переменные описываются внутри какого-то. 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;
```

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

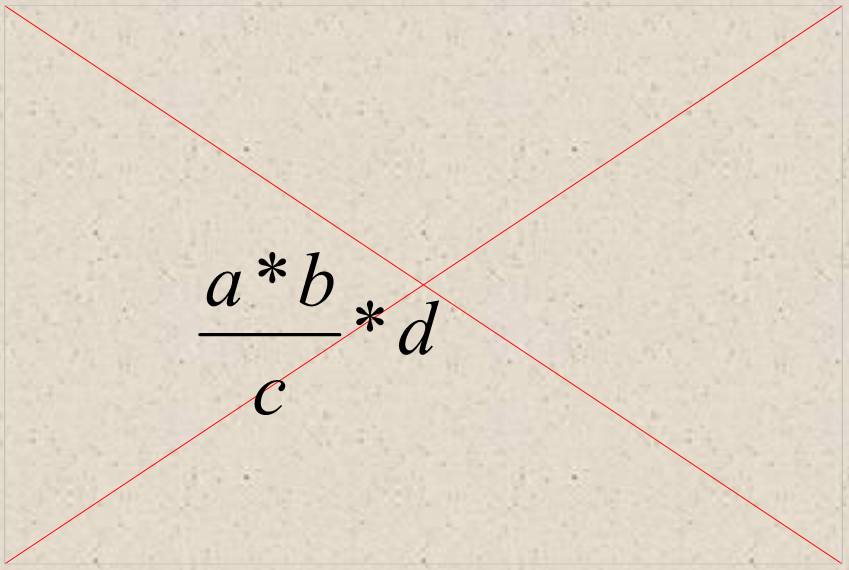
- a <= b + 2

результат имеет логический тип

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)))



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
```

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
```

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( x / y );          // Результат 2

             Console.WriteLine( x / z );         // Результат 2,75

            Console.WriteLine( x % y );          // Результат 3
}}}
```

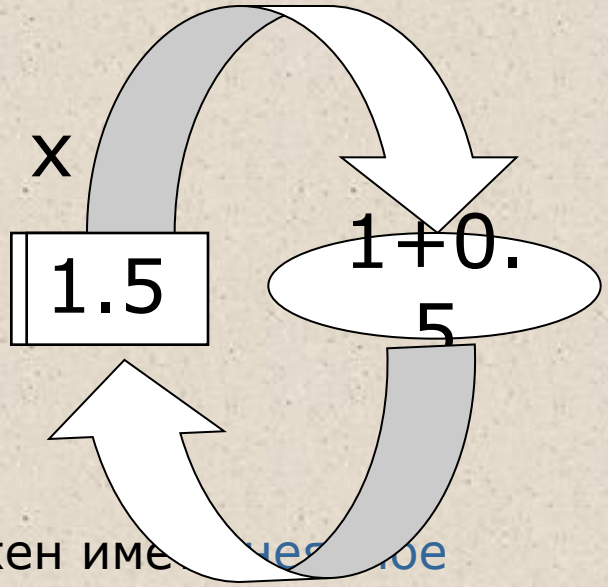
Операция присваивания

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 целое 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

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 );
      }
   }
}
```

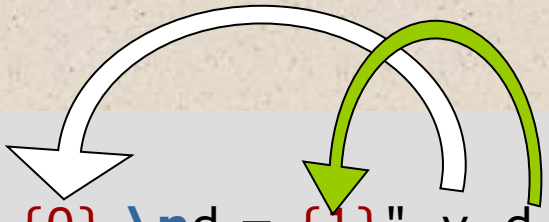
Результат работы программы:
3 y = 4,12
d = 600 s = Вася

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";

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

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



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";

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

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

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

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

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

```
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 );
    }
  }
}
```

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
- 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.