Основы языка программирования C#

(лекция 1. часть 1)

Коломеец Андрей Олегович

кандидат технических наук,
доцент кафедры «Информационные технологии на транспорте»,
старший научный сотрудник НИЛ «Физические методы контроля качества»

Основные критерии качества программы

- надежность
- возможность точно планировать производство и сопровождение

Для достижения этих целей программа должна:

- иметь простую структуру
- быть хорошо читаемой
- быть легко модифицируемой

Парадигмы программирования

- **Парадигма** – способ организации программы, принцип её построения. Наиболее распространёнными являются процедурная и объектно-ориентированная парадигмы. Они различаются способом декомпозиции, положенным в основу при создании программы.

- **Процедурная декомпозиция** состоит в том, что задача, реализуемая программой, делится на подзадачи, а они, в свою очередь – на более мелкие этапы, то есть выполняется пошаговая детализация алгоритма решения задачи.

- **Объектно-ориентированная декомпозиция** предполагает разбиение предметной области на объекты и реализацию этих объектов и их взаимосвязей в виде программы.

- Кроме этого существует **функциональная** и **логическая** парадигмы.

Первый взгляд на классы

- Понятие **класс** аналогично обыденному смыслу этого слова в контексте «класс членистоногих», «класс задач».

- Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых **экземплярами класса (объектами).**

- Класс содержит **данные**, задающие свойства объектов класса, и **функции (методы)**, определяющие их поведение.

- Все классы .NET имеют одного общего предка — класс object, и организованы в единую иерархическую структуру.

- Классы логически сгруппированы в так называемые **пространства имен**, которые служат для упорядочивания имен классов и предотвращения их конфликтов: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными.

Состав языка C#

- **Символы**
  - буквы: A-Z, a-z, _, буквы нац. алфавитов
  - цифры: 0-9, A-F
  - спец. символы: +, *, {, @ …
  - пробельные символы
- **Лексемы**
  - константы: 2 0.11 “Вася”
  - имена: Vasia a _11
  - ключевые слова: double do if
  - знаки операций: + - =
  - разделители ; [ ] ,
- **Выражения**
  - выражение – это правило вычисления значения: a + b
- **Операторы**
  - исполняемые: c = a + b
  - описания: double a, b;

Имена (идентификаторы)

- имя должно начинаться с буквы или со знака _ ;
- 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 множественного выборы слово **switch**.
- **Знак операции** — один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.
  - Например, сложение **+**, деление **/**, сложное присваивание **%=**.
- 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 языка C#

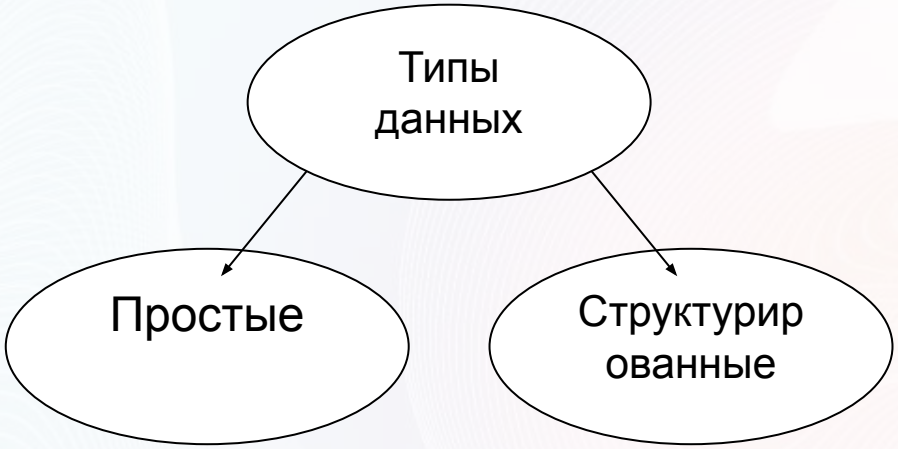
abstract as base bool break byte case catch
char checked class const continue decimal
default delegate do double else enum event explicit
extern false finally fixed float for foreach goto
if implicit in int interface internal is lock long
namespace new null object operator out override
params private 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

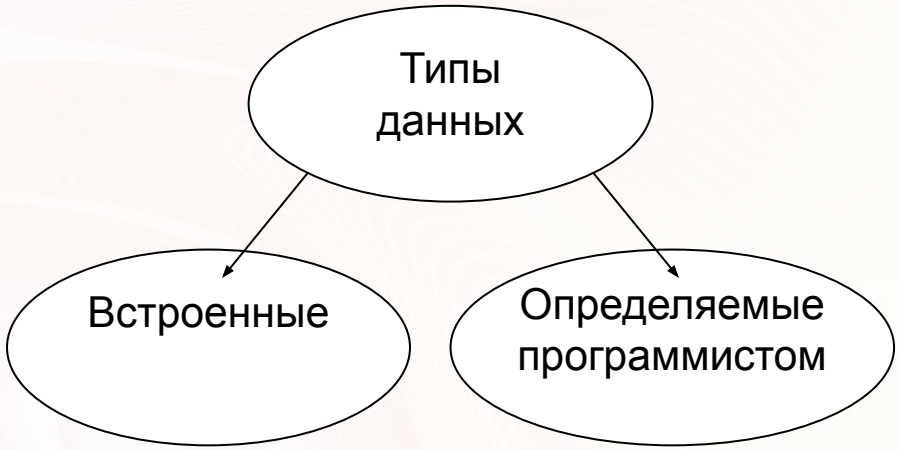
Концепция типа данных

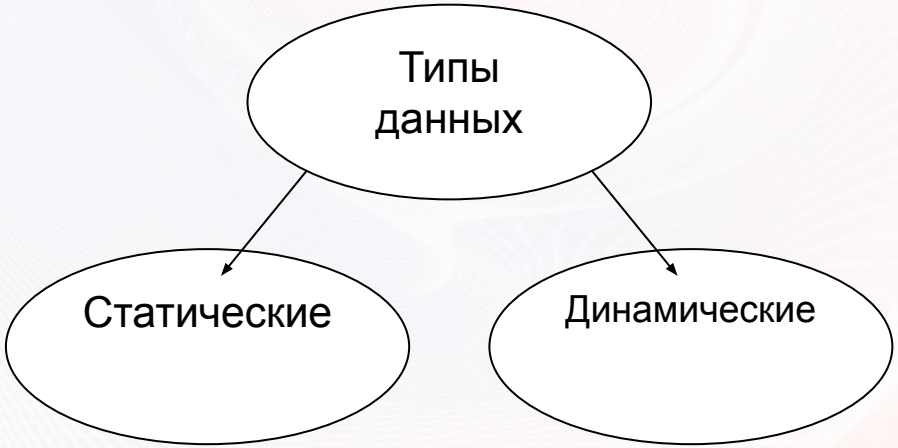
Тип данных определяет:

- **Внутреннее представление данных –**
  множество их возможных значений
- **Допустимые действия над ними –**
  операции и функции

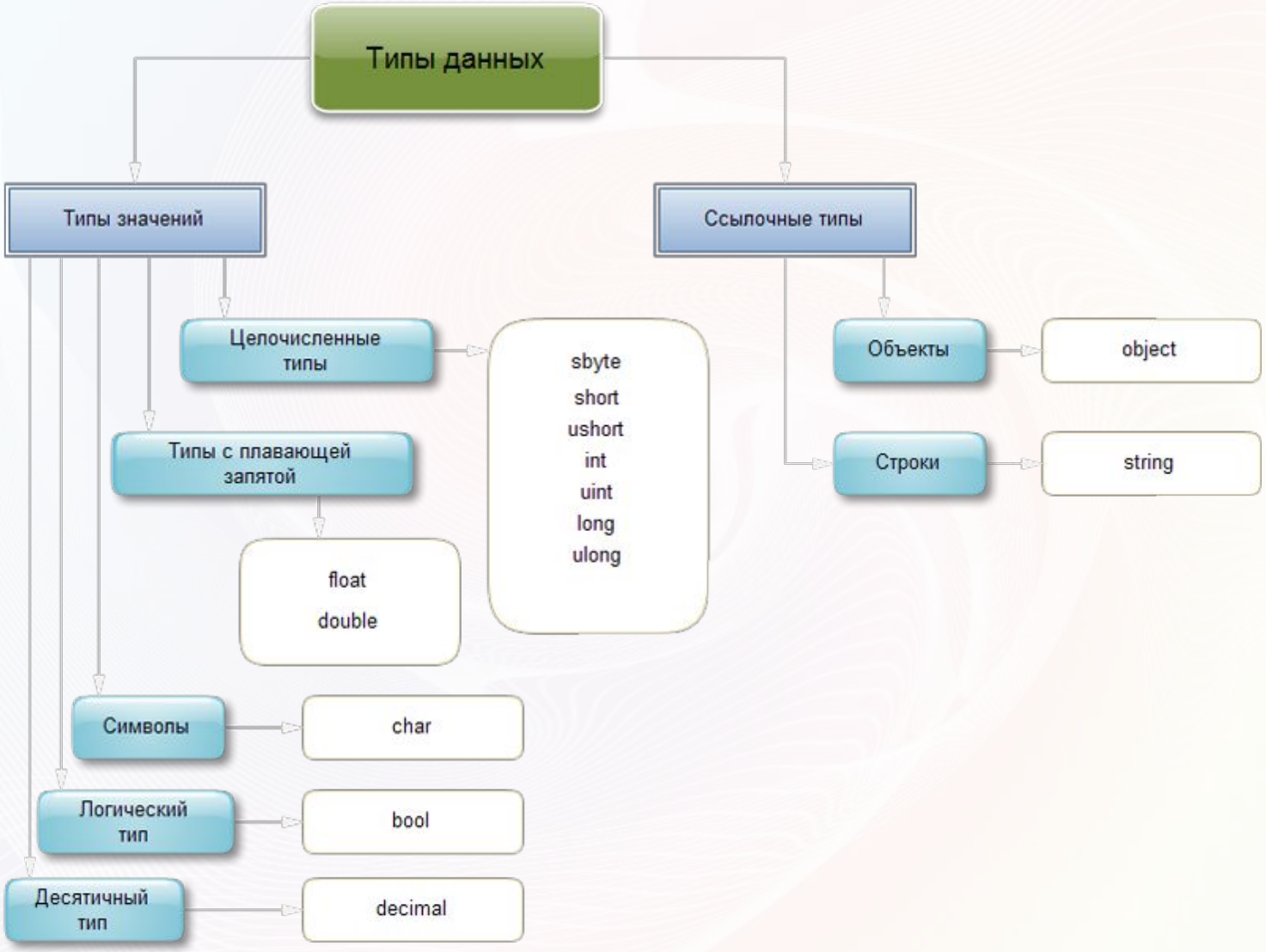
Различные классификации типов данных



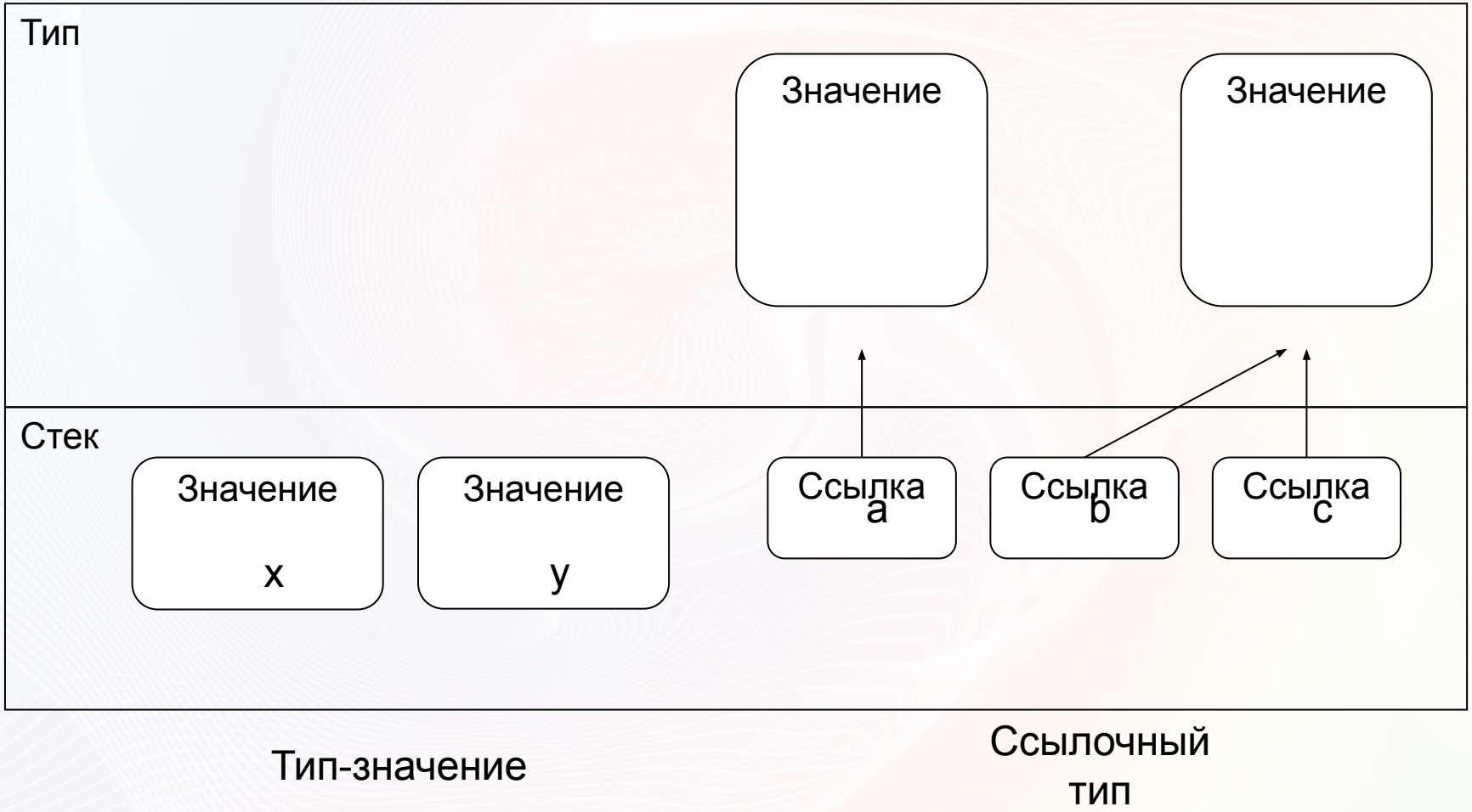




Классификация типов данных C#



Хранение в памяти величин значимого и ссылочного типа



Встроенные типы данных C#

Название	Ключевое слово	Тип .NET	Описание	Размер, бит
Булевский	**bool**	**Boolean**		
Целые	**sbyte**	**SByte**	знаковое	8
	byte	**Byte**	беззнаковое	8
	short	**Int16**	знаковое	16
	ushort	**UInt16**	беззнаковое	16
	int	**Int32**	знаковое	32
	uint	**UInt32**	беззнаковое	32
	long	**Int64**	знаковое	64
	ulong	**UInt64**	беззнаковое	64

Встроенные типы данных C#

Название	Ключевое слово	Тип .NET	Описание	Размер, бит
Символьный	**char**	**Char**	символ Unicode	16
Вещественные	**float**	**Single**	7 цифр	32
	double	**Double**	15-16 цифр	64
Финансовый	**decimal**	**Decimal**	28-29 цифр	128
Строковый	**string**	**String**	строка из символов Unicode	
object	**object**	**Object**	всеобщий предок	

Математические функции (класс Math)

- Для выполнения различных математических операций в библиотеке классов .NET предназначен класс **Math**.
- Содержит определение тригонометрических функций, математических операций, константы и прочее…
- Пример:

```
//код вашей программы
Console.Write("Введите радиус круга R = ");
double radius = Double.Parse(Console.ReadLine());
Console.WriteLine();

double area = Math.PI * Math.Pow(radius, 2);

Console.WriteLine("Площадь круга с радиусом {0} S = {1} ", radius, Math.Ceiling(area));
Console.ReadKey();
```

Результат программы:

```
C:\Users\Andr\Documents\SharpDevelop Projects\lab1\lab1\bin\Debug\lab1.ex
Введите радиус круга R = 12
Площадь круга с радиусом 12 S = 453
```

Математические функции (класс Math)

Имя	Описание	Результат	Пояснения
Abs	Модуль	**перегружен**	|x| записывается как Abs(x)
Acos	Арккосинус	double	Acos(double x)
Asin	Арксинус	double	Asin(double x)
Atan	Арктангенс	double	Atan(double x)
Ceiling	Округление до большего целого	double	Ceiling(double x)
Cos	Косинус	double	Cos(double x)
E	База натурального логарифма (число e)	double	2,71828182845905
Exp	Экспонента	double	e^x записывается как Exp(x)
Floor	Округление до меньшего целого	double	Floor(double x)

Математические функции (класс Math)

Имя	Описание	Результат	Пояснения
Log	Натуральный логарифм	double	$\log_e x$ записывается как Log(x)
Log10	Десятичный логарифм	double	$\log_{10} x$ записывается как Log10(x)
Max	Максимум из двух чисел	**перегружен**	Max(x, y)
Min	Минимум из двух чисел	**перегружен**	Min(x, y)
PI	Значение числа ПИ	double	3,14159265358979
Pow	Возведение в степень	double	x^y записывается как Pow(x, y)
Round	Округление	**перегружен**	Round(3.1) даст в результате 3 Round (3.8) даст в результате 4
Sin	Синус	double	Sin(double x)
Sqrt	Квадратный корень	double	$\sqrt{x}$ записывается как Sqrt(x)
Tan	Тангенс	double	Tan(double x)

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 = 100;
double x, y;
char option = 'ю';
string str = "Привет, ФБИ!";
```

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

Область действия и время жизни переменных

- Переменные описываются внутри какого-л. блока (класса, метода или блока внутри метода)
  - **Блок** — это 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 i = 3;
double y = 4.12;
decimal d = 600.01m;
string str = "Привет, ФБИ!";
```

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).
- Инициализация локальных переменных возлагается на программиста. Рекомендуется всегда инициализировать переменные при описании.

Структура простейшей программы C#

```
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              //код вашей программы
11
12
13              Console.ReadKey();
14          }
15      }
16  }
```

Пример описания переменных

```
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              //код вашей программы
11              int i = 3;
12              double y = 4.12;
13              decimal d = 600.01m;
14              string str = "Привет, ФБИ!";
15
16              Console.ReadKey();
17          }
18      }
19  }
20
21
```

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

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

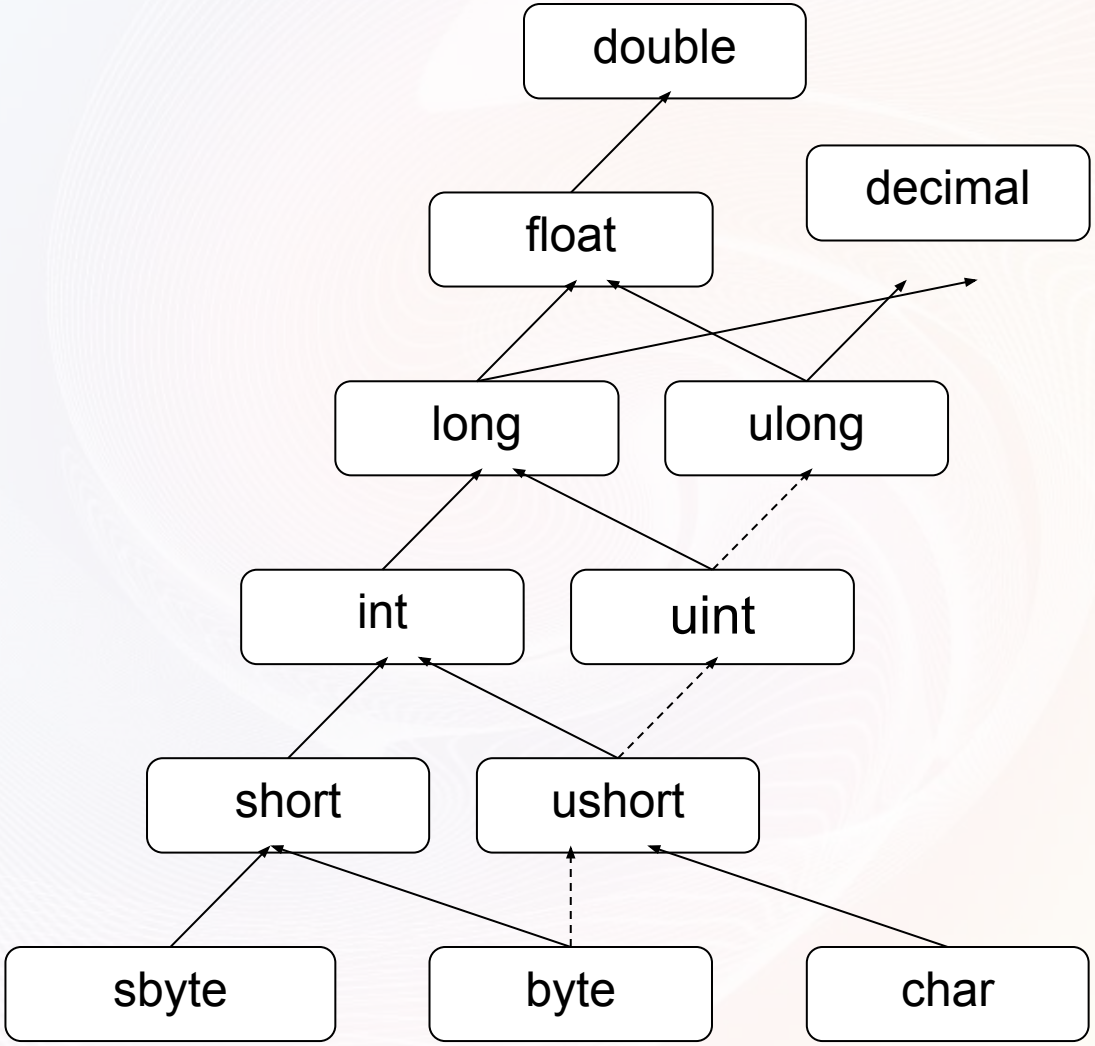
Приоритеты операция в C#

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 операнды, входящие в выражение, 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.
- 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**.

Неявные преобразования типов C#



Явные преобразования типов C#

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

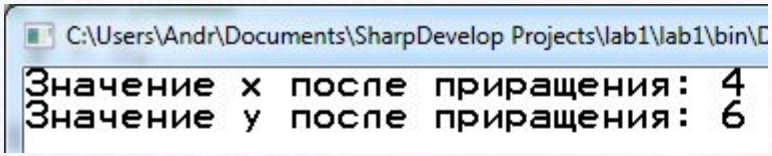
Введение в исключения

- 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.

Операция инкремент

```
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              //код вашей программы
11              int x = 3, y = 5;
12              x++;
13              Console.WriteLine("Значение x после приращения: "+x.ToString());
14              y++;
15              Console.WriteLine("Значение y после приращения: "+y.ToString());
16
17              Console.ReadKey();
18          }
19      }
20  }
```

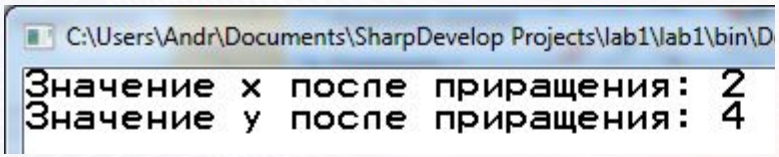
Результат программы:



Операция декремент

```
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              //код вашей программы
11              int x = 3, y = 5;
12              x--;
13              Console.WriteLine("Значение x после приращения: "+x.ToString());
14              y--;
15              Console.WriteLine("Значение y после приращения: "+y.ToString());
16
17              Console.ReadKey();
18          }
19      }
20  }
21
22
```

Результат программы:



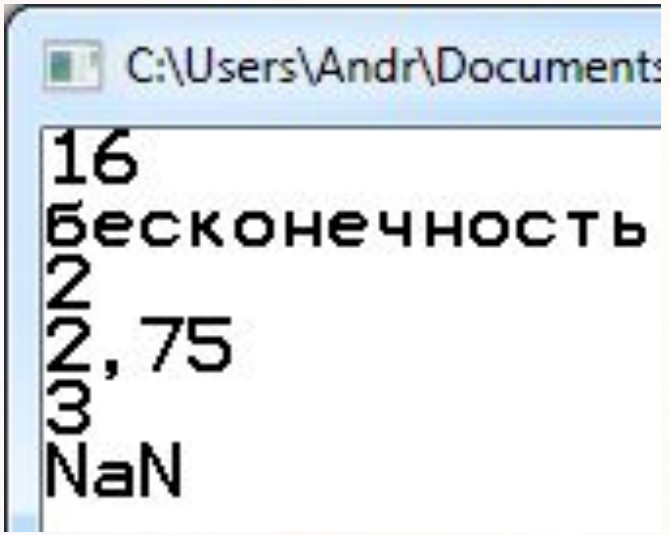
Умножение (деление)

- *Операция умножения* (*) 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

Пример

```
//код вашей программы
int x = 11, y = 4;
float z = 4;
Console.WriteLine(z*y);
Console.WriteLine(z*1e308);
Console.WriteLine(x/y);
Console.WriteLine(x/z);
Console.WriteLine(x%y);
Console.WriteLine(1e-324/1e-324);
```

Результат программы:



Операции отношения и проверки на равенство

- **Операции отношения** (<, <=, >, >=, ==, !=) 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

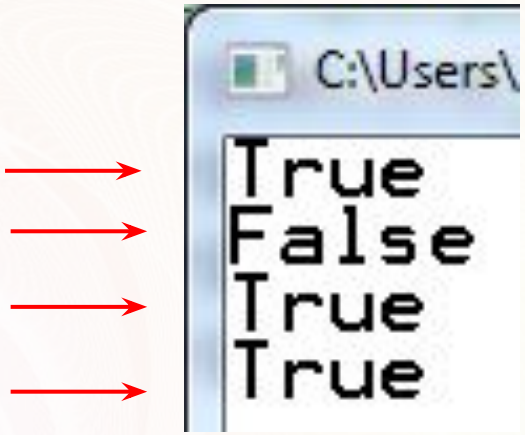
x <= y -- true, если x меньше или равно y, иначе false

x >= y -- true, если x больше или равно y, иначе false

Логические операции

```
//код вашей программы
Console.WriteLine(true && true);
Console.WriteLine(true && false);
Console.WriteLine(true || true);
Console.WriteLine(true || false);
```

Результат программы:



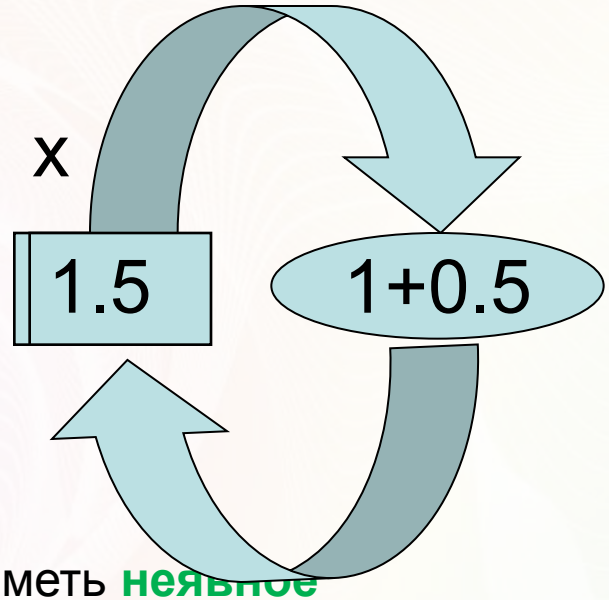
Операция присваивания

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 **неявное 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;

и т.п.

Ввод с консоли

Ввод данных с клавиатуры C#:

имя_переменной=тип.Parse(Console.ReadLine());

```
x=int.Parse(Console.ReadLine()); //ввод целого числа
y=float.Parse(Console.ReadLine()); //ввод десятичного числа
s=Console.ReadLine(); //ввод строки
```

Ввод с консоли

```
1
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              int x; float y; string str;
11
12              x = int.Parse(Console.ReadLine());
13              y = float.Parse(Console.ReadLine());
14              str = Console.ReadLine();
15
16              Console.ReadKey();
17          }
18      }
19  }
```

Вывод на консоль

Вывод данных на экран C#:

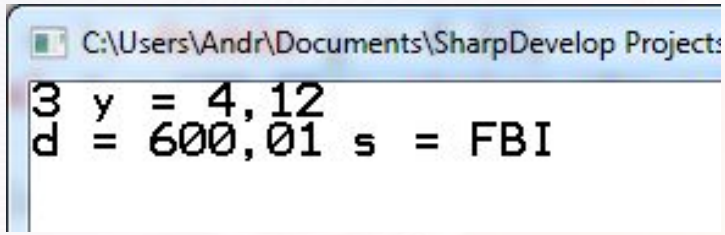
Console.WriteLine(имя_переменной);

```
Console.WriteLine(“Вывод строки”);
Console.WriteLine(x); //вывод значения X
Console.WriteLine(“Значение X:”+x.ToString());
```

Вывод на консоль

```
1
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              int i = 3; double y = 4.12;
11              decimal d = 600.01m; string str = "FBI";
12
13              Console.WriteLine(i.ToString()+" y = "+y.ToString());
14              Console.WriteLine("d = "+d.ToString() + " s = "+str.ToString());
15
16              Console.ReadKey();
17          }
18      }
19  }
```

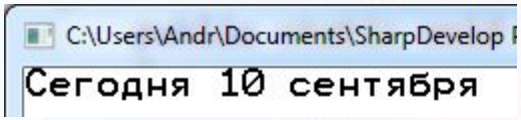
Результат программы:



Форматированный вывод

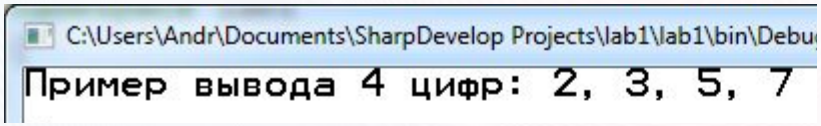
```
int day = 10;
Console.WriteLine("Сегодня {0} сентября", day);
```

Результат программы:



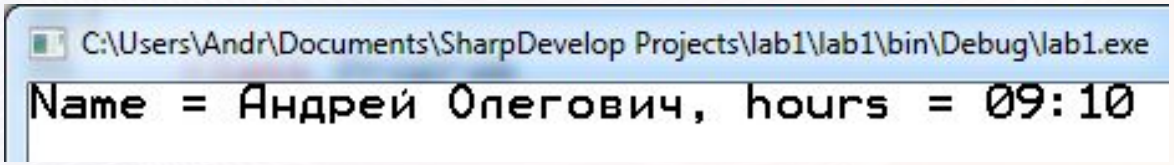

```
string primes;
primes = String.Format("Пример вывода 4 цифр: {0}, {1}, {2}, {3}",
           2, 3, 5, 7 );
Console.WriteLine(primes);
```

Результат программы:




```
string name = "Андрей Олегович";
string output_str;
output_str = String.Format("Name = {0}, hours = {1:hh:mm}", name, DateTime.Now);
Console.WriteLine(output_str);
```

Результат программы:



Пример консольного приложения в C#

```
2   using System;
3
4   namespace lab1
5   {
6       class Program
7       {
8           public static void Main(string[] args)
9           {
10              //код вашей программы
11              Console.WriteLine("Введите температуру по Фаренгейту");
12              double far = double.Parse(Console.ReadLine());
13
14              double cels = 5.0 / 9 * (far - 32);
15              Console.WriteLine("Температура по Фаренгейту {0} в градусах Цельсия равна {1:##.0}",
16                                far, cels);
17
18              Console.ReadKey();
19          }
20      }
21  }
```

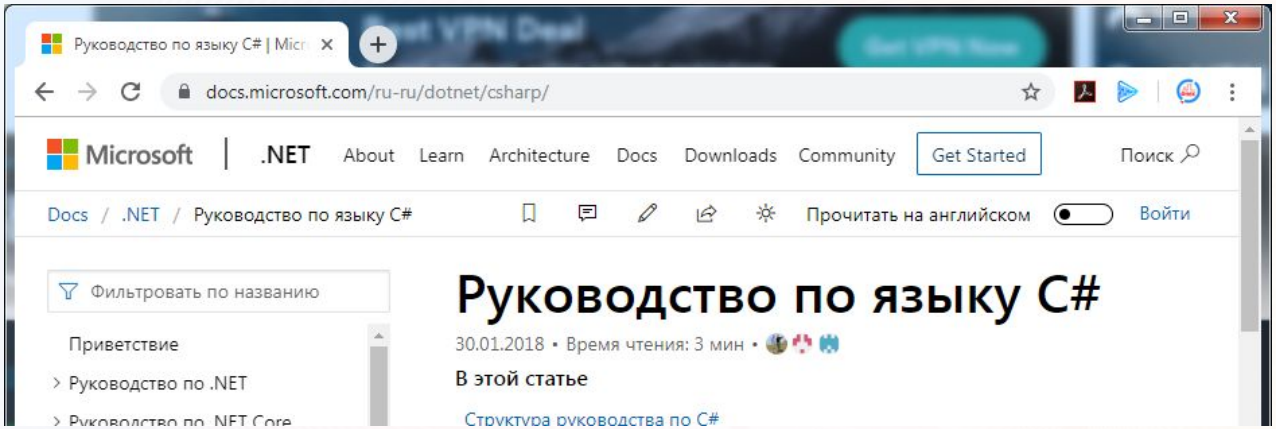
$$C = \frac{5}{9}(F - 32)$$

Результат программы:

```
C:\Users\Andr\Documents\SharpDevelop Projects\lab1\lab1\bin\Debug\lab1.exe
Введите температуру по Фаренгейту 220
Температура по Фаренгейту 220 в градусах Цельсия равна 104,4
```

Ссылки на учебный материал

Сайт MSDN (C#): https://docs.microsoft.com/ru-ru/dotnet/csharp/



Сайт «Профессор Веб»: https://professorweb.ru/