Язык программирования C#

Введение

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 ее построения. Наиболее распространенными 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 того, существуют *функциональная* и *логическая* парадигмы.

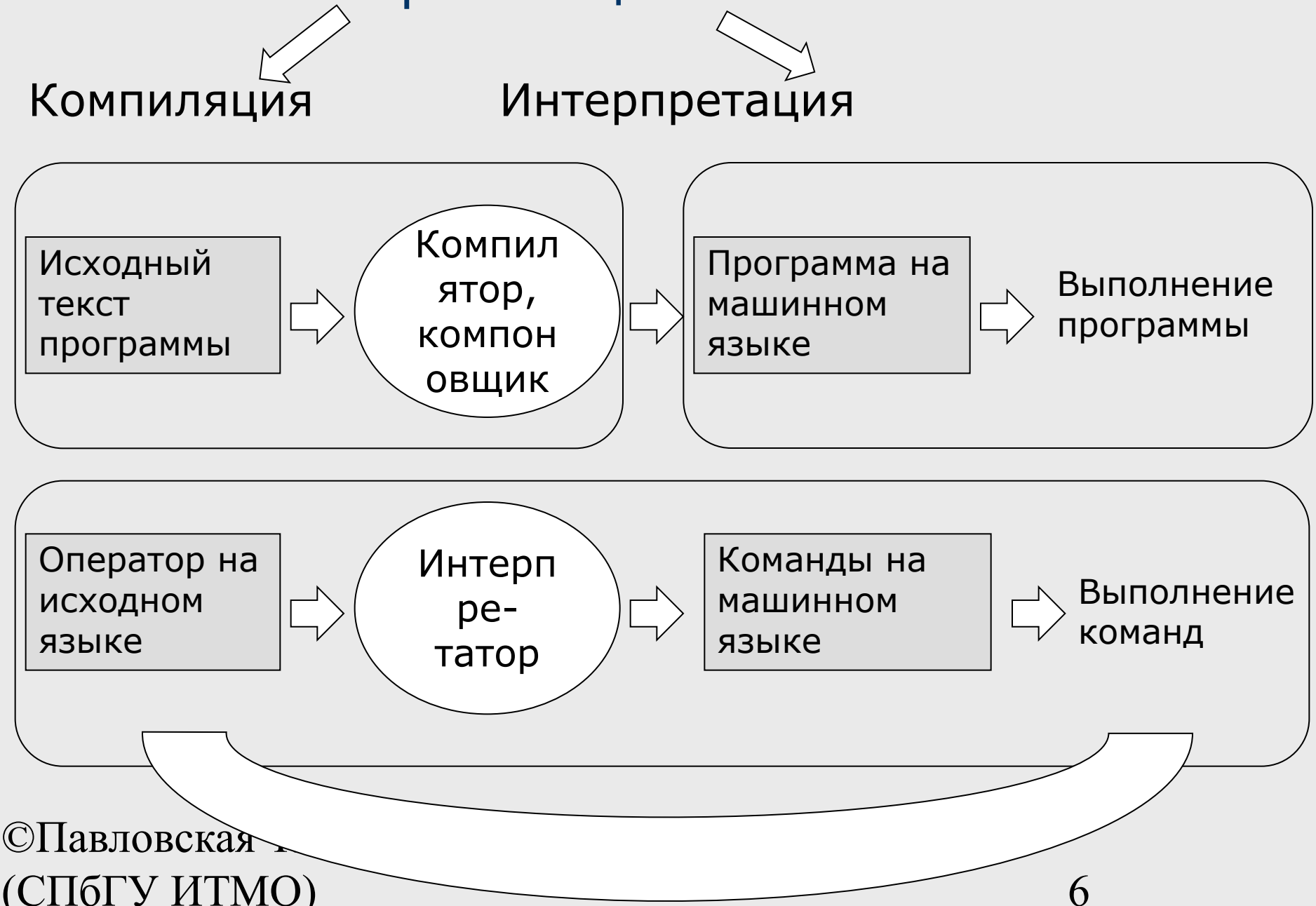
Первый взгляд на классы

- 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

Трансляция


Компиляция
Интерпретация
Исходный текст программы
Компилятор, компоновщик
Программа на машинном языке
Выполнение программы
Оператор на исходном языке
Интерпре-татор
Команды на машинном языке
Выполнение команд

Гибридная схема трансляции

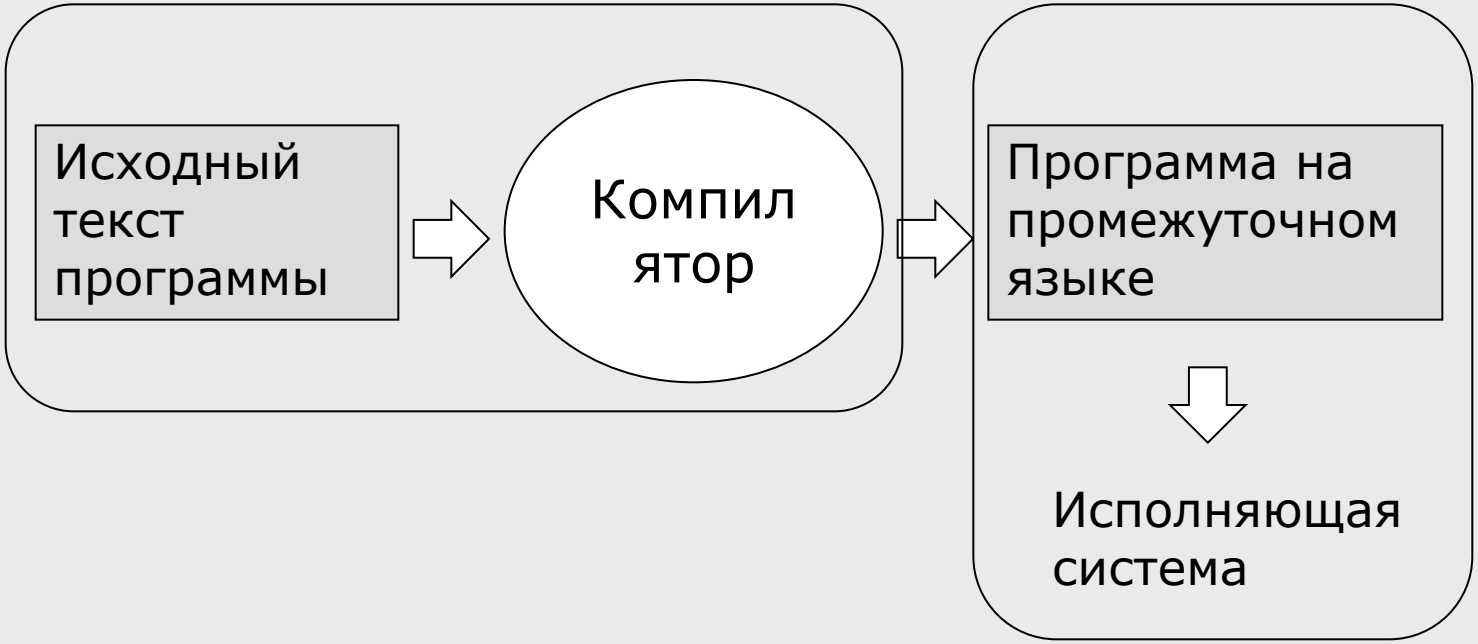
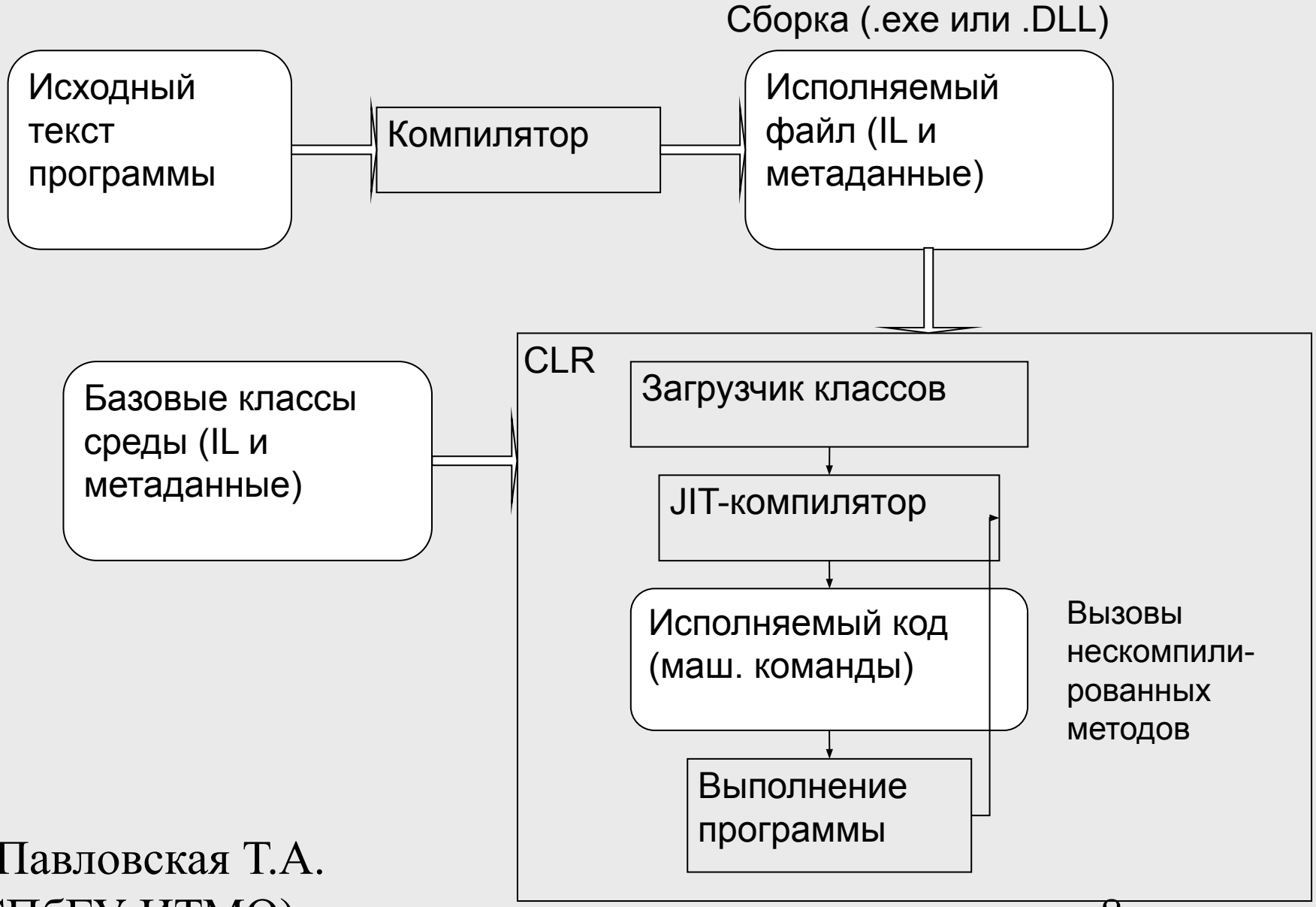


Схема выполнения программы в .NET



Основные понятия языка

Состав языка

Состав языка

- **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 - правило вычисления значения: a + b
- **Операторы**
  - исполняемые: c = a + b;
  - описания: double a, b;

Константы (литералы) C#

Вид	Примеры
Булевские	*true* *false*
Целые дес.	8 199226 0Lu
шестн.	0xA 0x1B8 0X00FFL
Веществ. с тчк	5.7 .001f 35m
с порядком	0.2E6 .11e–3 5E10
Символьные	'A' '\x74' '\0' '\uA81B'
Строковые	"Здесь был Vasia"
	"\t3начение r=\xF5\n"
	"Здесь был \u0056\u0061"
	@"C:\temp\file1.txt"
Константа null	null

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, 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 C#

```
abstract as base  bool  break byte  case  catch
char  checked  class const continue decimal
default  delegate do doubleelse  enum  event
explicit externfalse finally  fixed float for
foreach  goto  if implicit in intinterface
internal is lock  long  namespacenewnull  object
operator outoverride paramsprivate  protected
publicreadonly refreturnsbyte sealedshort sizeof
stackalloc  staticstringstructswitchthis  throw
true  trytypeofuint     ulong uncheckedunsafe
ushortusing virtual  void  volatile while
```

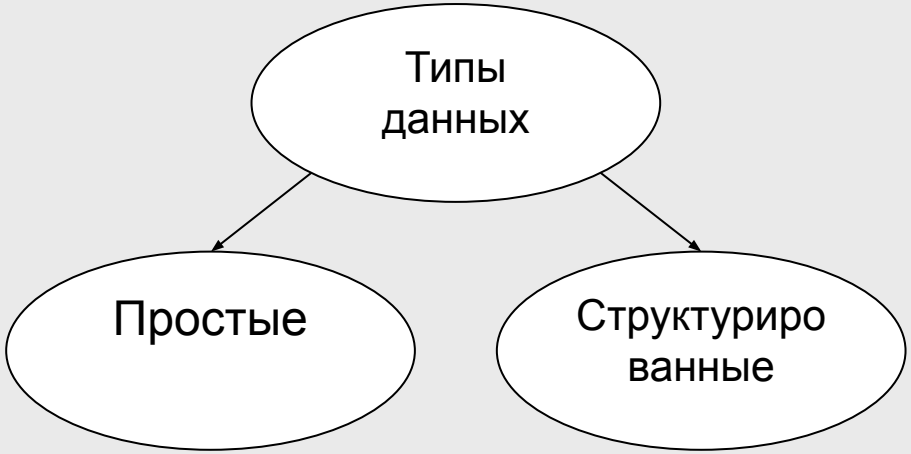
Типы данных

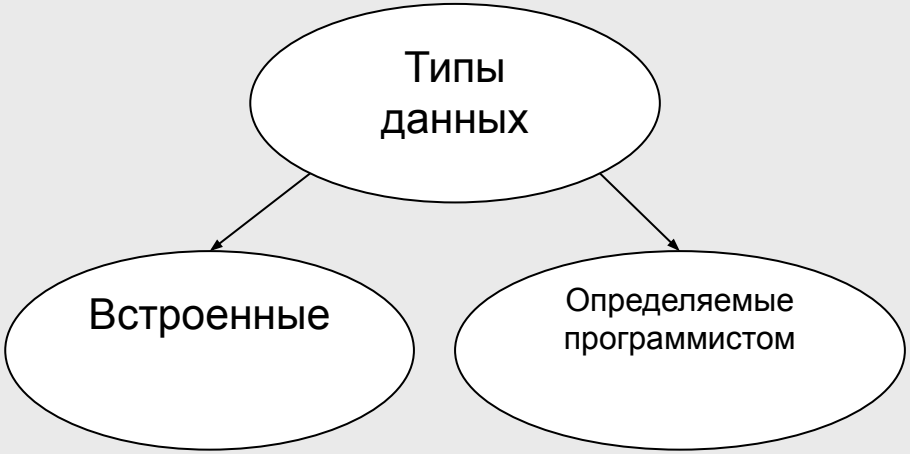
Концепция типа данных

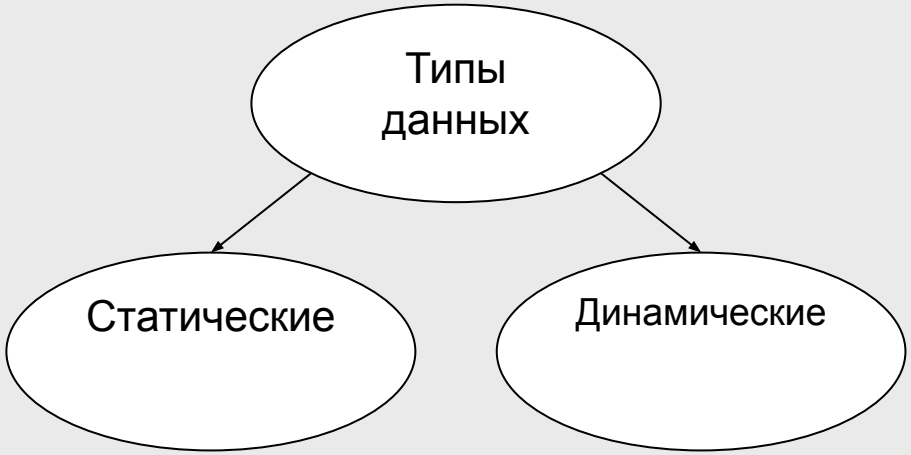
Тип данных определяет:

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

Различные классификации типов данных

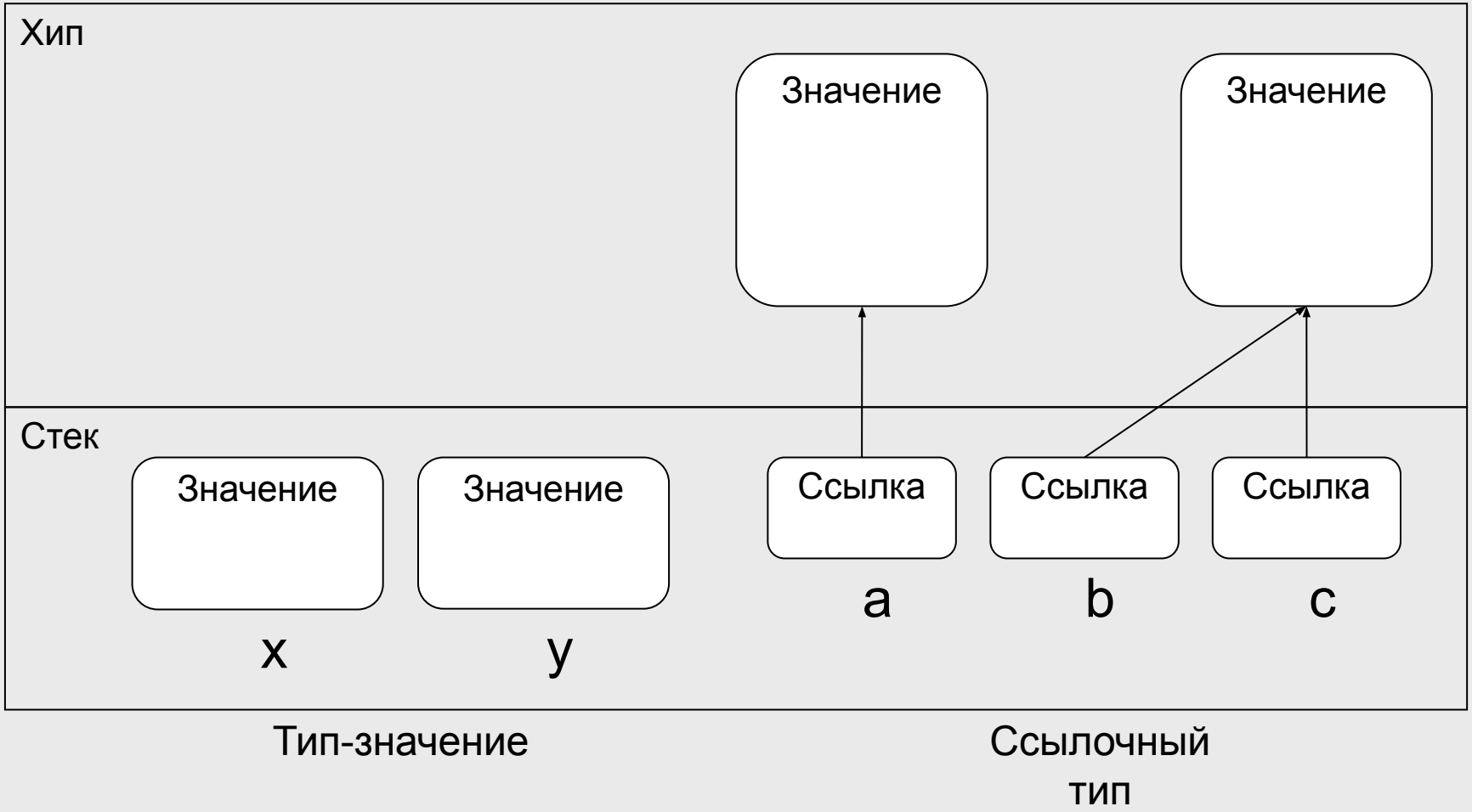






Основная классификация типов C#

Хранение в памяти величин значимого и ссылочного типа



Встроенные типы данных C#

Логический и целые

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
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`	$\approx(-2\cdot10^9 — 2\cdot10^9)$	знаковое	32
	uint	`UInt32`	$\approx(0 — 4\cdot10^9)$	беззнаковое	32
	long	`Int64`	$\approx(-9\cdot10^{18} — 9\cdot10^{18})$	знаковое	64
	ulong	`UInt64`	$\approx(0— 18\cdot10^{18})$	беззнаковое	64

Остальные

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
Символьный	char	**Char**	U+0000 — U+ffff	символ Unicode	16
Вещественные	float	**Single**	$1.5 10^{-45}$ — $3.4 10^{38}$	7 цифр	32
	double	**Double**	$5.0 10^{-324}$ — $1.7 10^{308}$	15-16 цифр	64
Финансовый	decimal	**Decimal**	$1.0 10^{-28}$ — $7.9 10^{28}$	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 как `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 как `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 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 π	`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
`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 как `Sqrt(x)`
`Tan`	Тангенс	`double`	`Tan(double x)`
`Tanh`	Гиперболический тангенс	`double`	`Tanh(double x)`

Линейные программы

Структура простейшей программы на C#

```
using System;
namespace A
{
    class Class1
    {
     static void Main()
        {

            // описания и операторы
        }

     // 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-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).

```
intnumber = 100;
   floatx    = 0.02;
   charoption = ’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).
- Инициализация локальных переменных возлагается на программиста. Рекомендуется всегда 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;
```

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

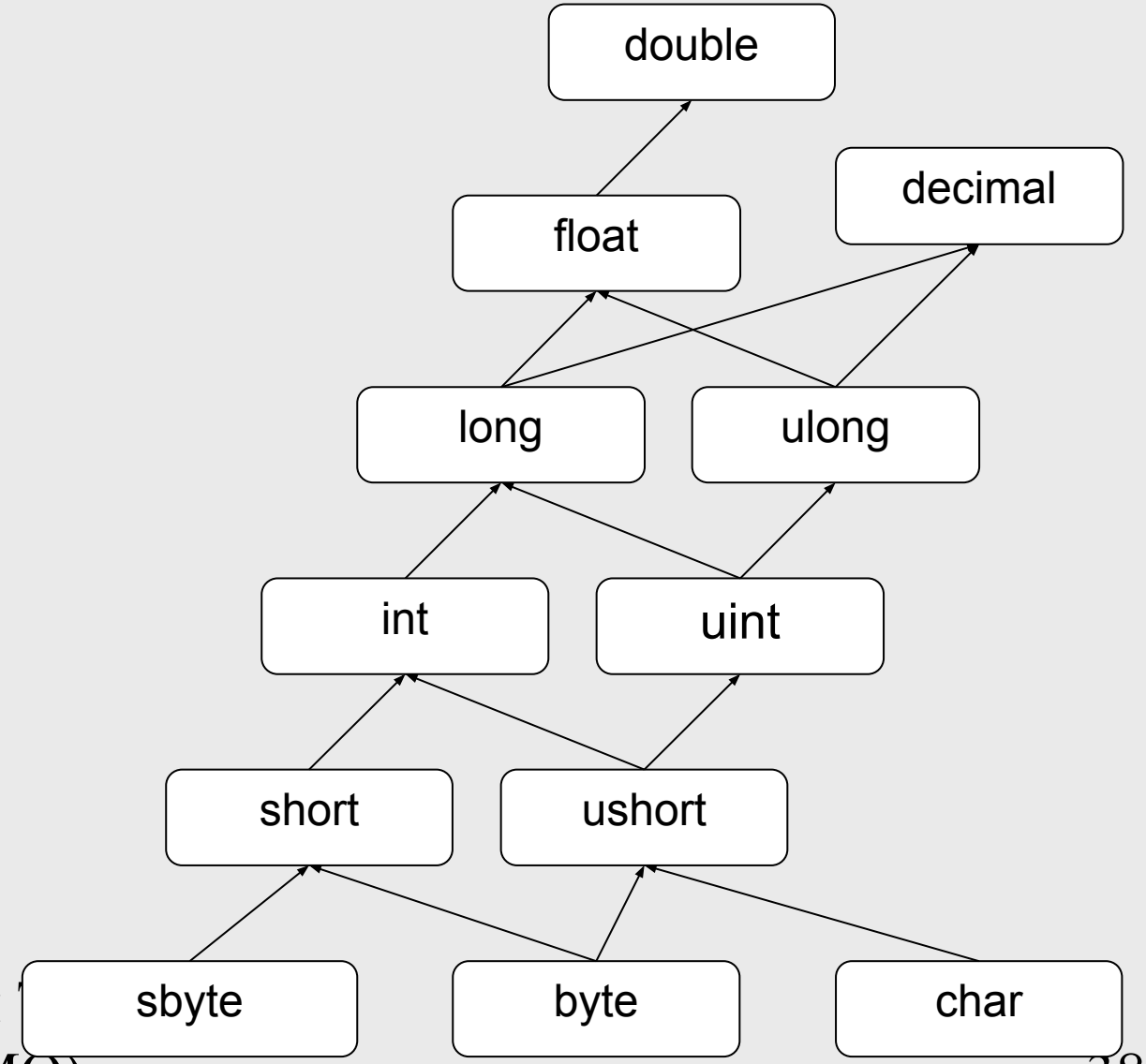
Приоритеты операций C#

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 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 арифметические преобразования типов в C#



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.

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:
```

Операция new

Операция new служит для создания нового объекта. Формат операции:

new тип ([аргументы])

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

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

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
}}}
```

Явное преобразование типа

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

Умножение

- *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

Результаты вещественного умножения

*	+y	-y	+0	-0	+∞	-∞	NaN
+x	+z	-z	+0	-0	+∞	-∞	NaN
-x	-z	+z	-0	+0	-∞	+∞	NaN
+0	+0	-0	+0	-0	NaN	NaN	NaN
-0	-0	+0	-0	+0	NaN	NaN	NaN
+∞	+∞	-∞	NaN	NaN	+∞	-∞	NaN
-∞	-∞	+∞	NaN	NaN	-∞	+∞	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Пример

```
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

- *Операции сдвига* (<< и >>) применяются к целочисленным операндам. Они 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
        }
    }
}
```

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

Условные логические операции

```
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
        }
    }
}
```

Условная операция

- **операнд_1 ? операнд_2 : операнд_3**

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

Если результат вычисления первого операнда равен true, то результатом будет значение второго операнда, иначе — третьего операнда.

```
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
 }}}
```

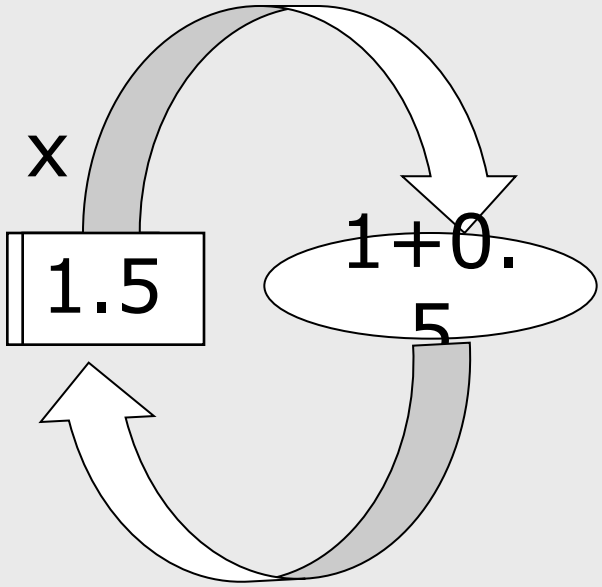
Операция присваивания

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 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;
- a <<= 2; соответствует a = a << 2;

и т.п.

Ввод-вывод в C#

Вывод на консоль

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

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

      }
   }
}
```

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

Ввод с консоли

```
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.
      }
   }
}
```

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" );
        string  buf  = Console.ReadLine();
        double fahr = Convert.ToDouble( buf );

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

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