Язык программирования C#

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

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
```

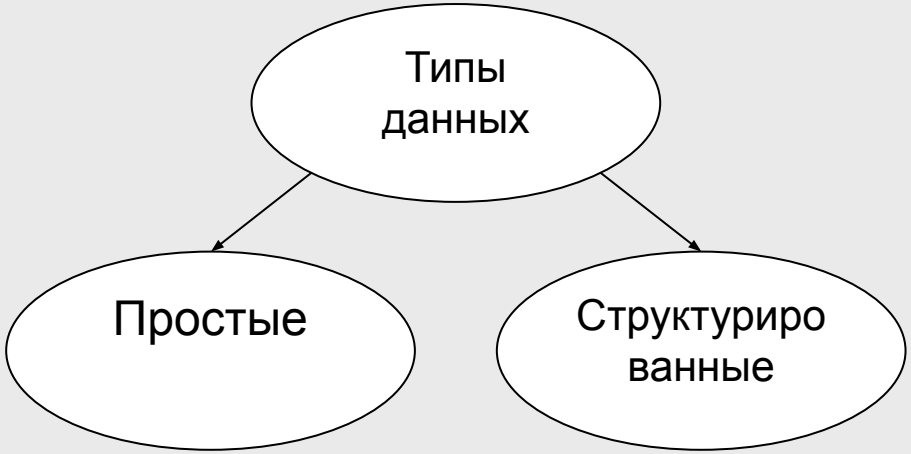
Типы данных

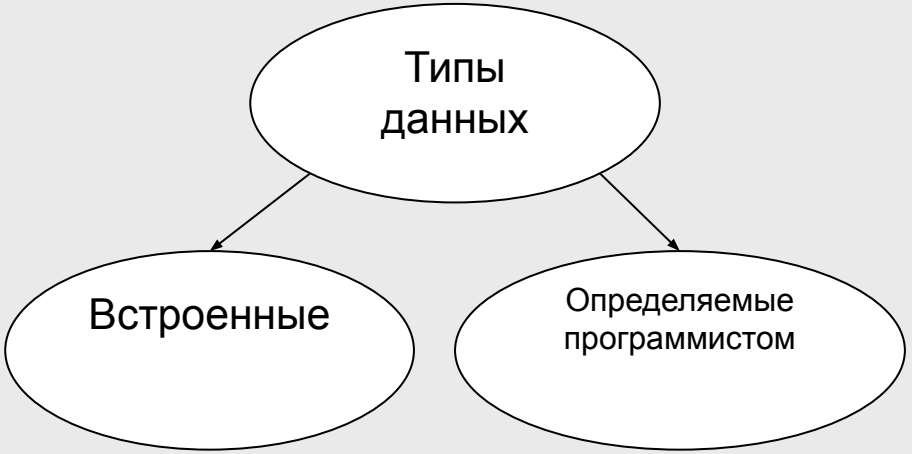
Концепция типа данных

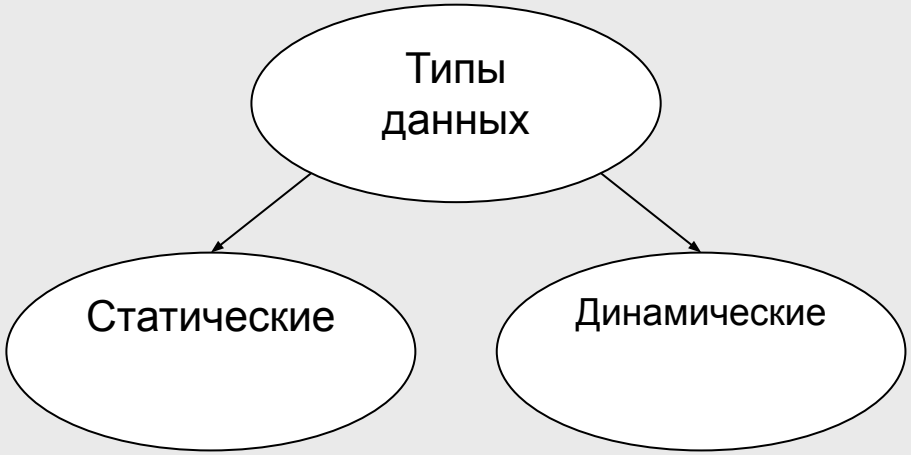
Тип данных определяет:

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

Различные классификации типов данных







Основная классификация типов C#

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`	$\approx(-2 10^9 — 2 10^9)$	знаковое	32
	uint	`UInt32`	$\approx(0 — 4 10^9)$	беззнаковое	32
	long	`Int64`	$\approx(-9 10^{18} — 9 10^{18})$	знаковое	64
	ulong	`UInt64`	$\approx(0— 18 10^{18})$	беззнаковое	64

Остальные

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).

```
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).
- 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";

      }
   }
}
```

- При объявлении переменной ее можно инициализировать (присвоить ей начальное значение), а затем в любой момент ей можно присвоить новое значение, которое заменит собой предыдущее.

```
static void Main()
{
    int i=10; //объявление и инициализация целочисленной переменной i
     Console.WriteLine(i); //просмотр значения переменной
   i=100;      //изменение значение переменной
     Console.WriteLine(i);
}
```

- В языках предыдущего поколения переменные можно было использовать без инициализации. Это могло привести к множеству проблем и долгому поиску ошибок. В языке C# требуется, чтобы переменные были явно проинициализированы до их использования. Проверим этот факт на примере.

```
static void Main()
{
    int i; //объявление переменной без инициализации
    Console.WriteLine(i);  //просмотр значения переменной
}
```

При попытке скомпилировать этот пример в списке ошибок будет выведено следующее сообщение: ***«Использование локальной переменной i, которой не присвоено значение»***.

Инициализировать каждую переменную сразу при объявлении необязательно, но необходимо присвоить ей значение до того, как она будет использована.

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;
```

Константы бывают трех видов: литералы, типизированные константы и перечисления. В операторе присваивания:

x=32;

число 32 является литеральной константой. Его значение всегда равно 32 и его нельзя изменить.

Типизированные константы именуют постоянные значения. Объявление типизированной константы происходит следующим образом:

const <тип> <идентификатор> = <значение>;

Рассмотрим пример:

```
static void Main()
{
    const int i=10;  //объявление целочисленной константы i
   Console.WriteLine(i); //просмотр значения константы
   i=100;              //ошибка – недопустимо изменять значение константы
    Console.WriteLine(i);
}
```

Перечисления (enumerations) являются альтернативой константам.

Перечисление - это особый размерный тип, состоящий из набора именованных констант (называемых списком перечисления).

Синтаксис объявления перечисления следующий:

[атрибуты] [модификаторы] enum <имя> [: базовый тип] {список-перечисления констант (через запятую)};

Замечание. Атрибуты и модификаторы являются необязательными элементами этой конструкции. Более подробные сведения о их можно найти в дополнительных источниках информации.

Базовый тип - это тип самого перечисления. Если не указать базовый тип, то по умолчанию будет использован тип int. В качестве базового типа можно выбрать любой целый тип, кроме char.

Пример использования перечислений

```
class Program
{
enum gradus:int
{
        min=0,
        krit=72,
        max=100, //1
}

static void Main()
{
   Console.WriteLine("минимальная температура=" + (int) gradus.min);
   Console.WriteLine("критическая температура=" + (int)gradus.krit);
   Console.WriteLine("максимальная температура=" + (int)gradus.max);
}
}
```

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

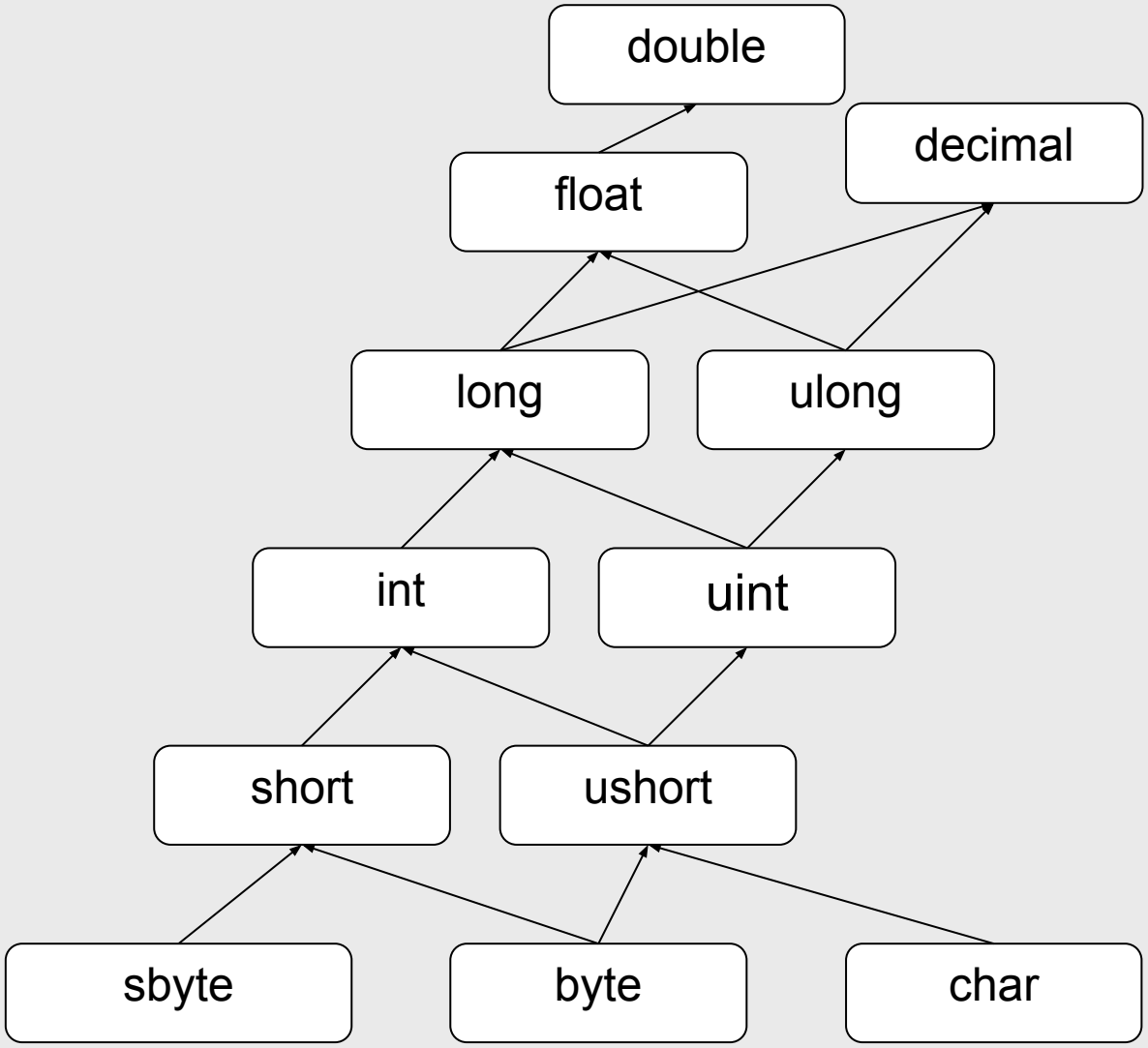
Приоритеты операций C#

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#



Ввод-вывод в C#

1. **Console.WriteLine(x);** *//на экран выводится значение идентификатора x*

2. **Console.WriteLine("x=" + x +"y=" + y);** */* на экран выводится строка, образованная последовательным слиянием строки "x=", значения x, строки "y=" и значения y */*

3. **Console.WriteLine("x={0} y={1}", x, y);** */* на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра {0} выводится значение x, а вместо {1} – значение y*/*

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

```
int i=3, j=4;
Console.WriteLine("{0} {1}", i, j);
```

При обращении к методу WriteLine через запятую перечисляются три аргумента: "{0} {1}", i, j. Первый аргумент "{0} {1}" определяет формат выходной строки. Следующие аргументы нумеруются с нуля, так переменная i имеет номер 0, j – номер 1. Значение переменной i будет помещено в выходную строку на место параметра {0}, а значение переменной j - на место параметра {1}. В результате на экран будет выведена строка: 3 4. Если мы обратимся к методу WriteLine следующим образом:

```
Console.WriteLine("{0} {1} {0)", j, i);
```

то на экран будет выведена строка: 4 3 4.

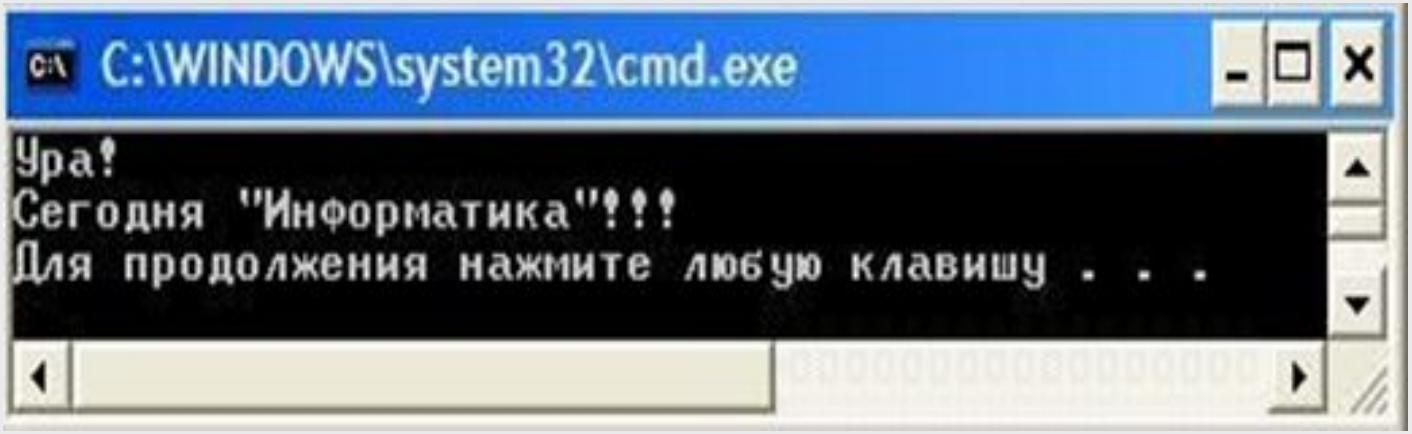
Использование управляющих последовательностей

Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой. Данная совокупность символов интерпретируется как одиночный символ и используется для представления кодов символов, не имеющих графического обозначения (например, символа перевода курсора на новую строку) или символов, имеющих специальное обозначение в символьных и строковых константах (например, апостроф).

Вид	**Наименование**
\a	Звуковой сигнал
\b	Возврат на шаг назад
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\’	Апостроф
\”	Кавычки

Пример:

```
static void Main()
{
                Console.WriteLine("Ура!\nСегодня    \"
Информатика\"!!!");
}
```



Задание. Измените программу так, чтобы все сообщение выводилось в одну строку, а после вывода сообщения раздавался звуковой сигнал.

Управление размером поля вывода:

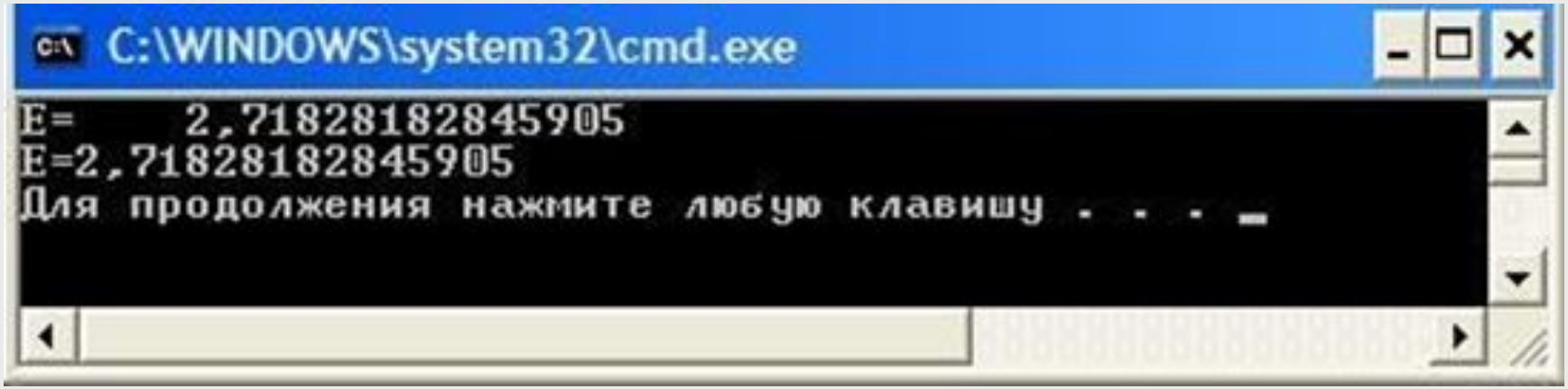
Первым аргументом WriteLine указывается строка вида {n, m} – где n определяет номер идентификатора из списка аргументов метода WriteLine, а m – количество позиций (размер поля вывода), отводимых под значение данного идентификатора.

При этом значение идентификатора выравнивается по правому краю.

Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций.

Пример

```
static void Main()
{
    double x= Math.E;
    Console.WriteLine("E={0,20}", x);
    Console.WriteLine("E={0,10}", x);
}
```

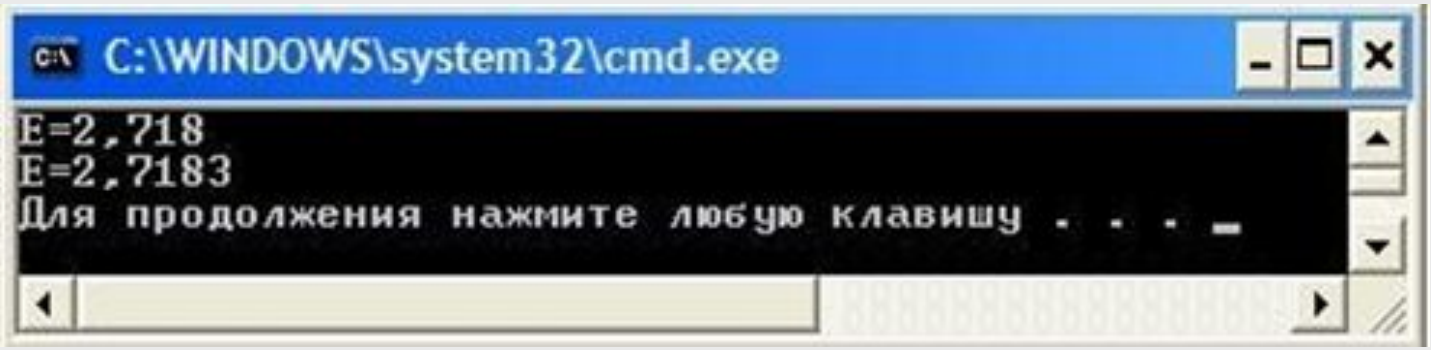


Управление размещением вещественных данных:

Первым аргументом WriteLine указывается строка вида {n: ##.###} – где n определяет номер идентификатора из списка аргументов метода WriteLine, а ##.### определяет формат вывода вещественного числа. В данном случае, под целую часть числа отводится две позиции, под дробную – три. Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций.

Пример

```
static void Main()
{
    double x= Math.E;
    Console.WriteLine("E={0:##.###}", x);
    Console.WriteLine("E={0:.####}", x);
 }
```



Задание. Измените программу так, чтобы число e выводилось на экран с точностью до 6 знаков после запятой.

Управление форматом числовых данных:

- Первым аргументом WriteLine указывается строка вида {n:<спецификатор>m} – где n определяет номер идентификатора из списка аргументов метода WriteLine, <спецификатор> - определяет формат данных, а m – количество позиций для дробной части значения идентификатора. В качестве спецификаторов могут использоваться следующие значения

Параметр	***Формат***	***Значение***
C или c	Денежный. По умолчанию ставит денежный знак, определенный текущими региональными настройками. В русской Windows это р.	Задается количество десятичных разрядов.
D или d	Целочисленный (используется только с целыми числами)	Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями
E или e	Экспоненциальное представление чисел	Задается количество символов после запятой. По умолчанию используется значение 6.
F или f	Представление чисел с фиксированной точкой	Задается количество символов после запятой
G или g	Общий формат (или экспоненциальный, или с фиксированной точкой)	Задается количество символов после запятой. По умолчанию выводится целая часть
N или n	Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами	Задается количество символов после запятой. По умолчанию – 2, если число целое, то ставятся нули
X или x	Шестнадцатеричный формат	
P или p	Процентный	

Пример:

```
static void Main()
{
   Console.WriteLine("C Format:{0,14:C} \t{0:C2}", 12345.678);
   Console.WriteLine("D Format:{0,14:D} \t{0:D6}", 123);
   Console.WriteLine("E Format:{0,14:E} \t{0:E8}", 12345.6789);
   Console.WriteLine("G Format:{0,14:G} \t{0:G10}", 12345.6789);
   Console.WriteLine("N Format:{0,14:N} \t{0:N4}", 12345.6789);
   Console.WriteLine("X Format:{0,14:X} ", 1234);
   Console.WriteLine("P Format:{0,14:P} ", 0.9);
}
```

Ввод данных

Для ввода данных обычно используется метод ReadLine, реализованный в классе Console. Данный метод в качестве результата возвращает строку, тип которой string.

Пример:

```
static void Main()
{
  string s = Console.ReadLine();
  Console.WriteLine(s);
}
```

Для того чтобы получить числовое значение, необходимо воспользоваться преобразованием данных. Пример:

```
static void Main()
{
  string s = Console.ReadLine();
   int x = int.Parse(s); //преобразование строки в число
  Console.WriteLine(x);
}
```

Сокращенный вариант:

```
static void Main()
{
  //преобразование введенной строки в число
  int x = int.Parse(Console.ReadLine());
  Console.WriteLine(x);
}
```

- Для преобразования строкового представления целого числа в тип int мы используем метод Parse(), который реализован для всех числовых типов данных. Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом float.Parse() или double.Parse(). В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается и генерируется исключение. Например, если входная строка имела неверный формат, то будет сгенерировано исключение System.FormatExeption.

- ***Задания.***
- Подумайте, какие еще исключения могут возникнуть при использовании метода Parse. Проверьте свои предположения на практике.
- Измените предыдущий фрагмент программы так, чтобы с клавиатуры вводилось вещественное число, а на экран это число выводилось с точностью до 3 знаков после запятой.

Преобразование в другие типы

```
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 );
    }
  }
}
```

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.

Операции

Инкремент и декремент

Эти операции имеют две формы записи — префиксную, когда операция записывается перед операндом, и постфиксную – операция записывается после операнда.

Префиксная операция инкремента (декремента) увеличивает (уменьшает) свой операнд и возвращает измененное значение как результат.

Постфиксные версии инкремента и декремента возвращают первоначальное значение операнда, а затем изменяют его.

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

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

1. Арифметическое отрицание (-) – меняет знак операнда на противоположный.
2. Логическое отрицание (!) – определяет операцию инверсии для логического типа. Рассмотрим эти операции на примере.

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

```
static void Main()
{
    int i = -4;
    byte j = 4;
    int a = (int)j;  //преобразование без потери точности
   byte b = (byte)i; //преобразование с потерей точности
    Console.WriteLine("{0} {1}", a, b);
      }
```

Результат работы программы:

4 252

Задание. Объясните, почему операция (byte)i вместо ожидаемого значения -4 дала нам в качестве результата значение 252.

Умножение (*), деление (/) и деление с остатком (%)

- Стандартная операция умножения определена для типов int, uint, long, ulong, float, double и decimal.
- К величинам других типов 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
}}}
```

Задания

1. Выполните фрагмент программы и объясните полученный результат:

```
double a=100, b=33;
Console.WriteLine(a/b);
double d=100/33;
Console.WriteLine(d);
```

2. Выясните, чему будет равен результат операции, и объясните, как получился данный результат:

а) 1.0/0; б) 1/0

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

Задание.

Выясните, чему равен результат данного выражения:

1) 10<25<30

2) true<false

И объясните, как получился данный результат.

Условные логические операции

Результат логической операции **И** имеет значение истина тогда и только тогда, когда оба операнда принимают значение истина.

Результат логической операции **ИЛИ** имеет значение истина тогда и только тогда, когда хотя бы один из операндов принимает значение истина.

```
static void Main()
 {
   Console.WriteLine("x     y     x и y    x или y");
   Console.WriteLine("{0} {1} {2} {3}", false, false, false&&false, false||false);
   Console.WriteLine("{0} {1} {2} {3}", false, true, false&&true, false||true);
   Console.WriteLine("{0} {1} {2} {3}", true, false, true&&false, true||false);
Console.WriteLine("{0} {1} {2} {3}", true, true, true&&true, true||true);
    }
```

Результат работы программы:

```
x     y     x и y     x или y
      False    False       False       False
False     True       False       True
True      False       False     True
True      True       True      True
```

Замечание. Фактически была построена таблица истинности для логических операций И и ИЛИ.

Условная операция

- **операнд_1 ? операнд_2 : операнд_3**

Операнд1 – это логическое выражение, которое оценивается с точки зрения его эквивалентности константам true и false

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
 }}}
```

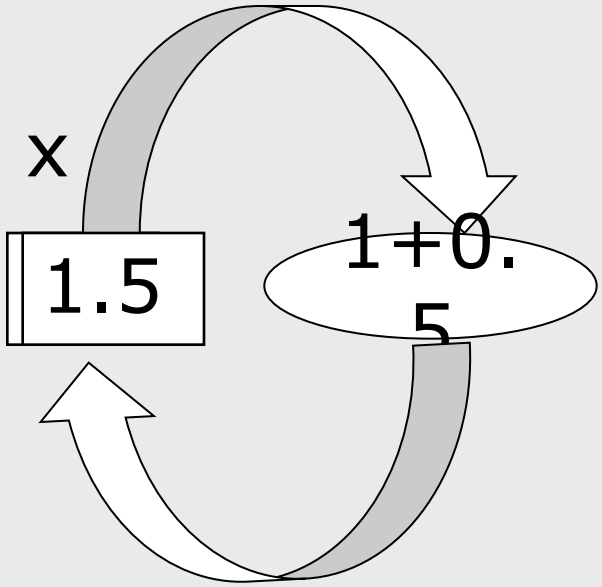
Операция присваивания

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.