САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

- **Сотавов Абакар Капланович**
- **Ассистент кафедры Информатики**(наб. канала Грибоедова, 30/32, ауд. 2038
- e-mail: sotavov@unecon.ru

- Материалы на сайте: http://de.unecon.ru/course/view.php?id=440

Переменные, операции, выражения

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.

Переменные

Переменная — это величина, которая во время работы программы может изменять св

Все переменные, используем
быть описаны.

Объявления переменных в C#
вид:

[модификаторы] <тип> <имя_переменной>;

неопределенный тип (var)
Например, объявление переменной:
var name = "Петров А.В.";
аналогично следующему объявлению:
string name = "Петров А.В.";

Для каждой переменной задается ее ***имя*** и ***тип***:

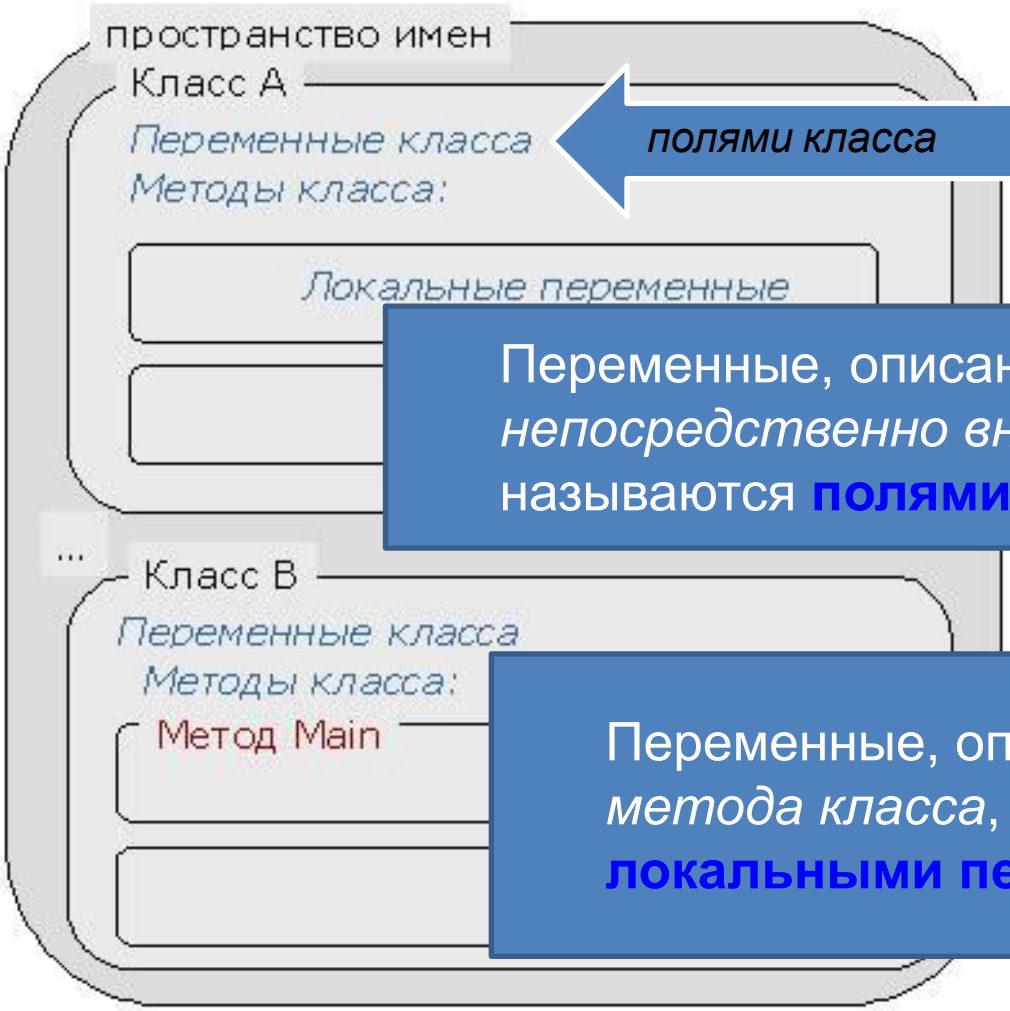
```
int a, b = 1;
float  x = 0.1, y=0.1f;
char    option;
```

```
int b = 1, a = 100;
```

```
int x = b * a + 25;
```

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

Общая структура программы на C#



полями класса

Переменные, описанные *непосредственно внутри класса*, называются **полями класса**.

Переменные, описанные *внутри метода класса*, называются **локальными переменными**.

Области видимости переменных

```
class X    // начало описания класса X
{
   int A;       // поле A класса X
   int B;       // поле B класса X

   void Y()     // ---------------------------------- метод Y класса X
   {
      int C;       // локальная переменная C, область действия - метод Y
      int A;       // локальная переменная A (НЕ конфликтует с полем A)

      {            // ============ вложенный блок 1 ============
         int D;       // локальная переменная D, область действия - этот блок
//       int A;       // недопустимо! Ошибка компиляции, конфликт с локальной
//                       переменной A
         C = B;       // присваивание переменной C поля B класса X (**)
         C = this.A; // присваивание переменной C поля A класса X (***)
      }            // ============ конец блока 1 ===============

      {            // ============ вложенный блок 2 ============
         int D;       // локальная переменная D, область действия - этот блок
      }            // ============ конец блока 2 ===============

   }            // ------------------------- конец описания метода Y класса X

}          // конец описания класса X
```

Области видимости переменных

```
using System;
namespace ConsoleApplication1
{    class Class1
    {    static void Main()
        {
            int     i = 3;
            double  y = 4.12;
            decimal d = 600m;
            string  s = "Вася";

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

Именованные константы

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

Это облегчает читабельность программы и внесение в нее изменений:

```
const int b = 1;
const float x = 0.1, y = 0.1f;
const int b = 1, a = 100;
const int x = b * a + 25;
```

Ссылочные переменые

операция new выделяет и инициализирует память для экземпляра любого заданного типа (для ссылочных переменных – в куче);

```
Box b1;              // объявление ссылочной переменной
b1 = new Box ();  // выделение памяти для объекта
```

```
Box b1 = new Box(); // объявление и инициализация
```

Операции и выражения

Операции и 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

Операции и выражения

Категория	Знак операции
Первичные	(), [], ++, --, new, typeof…
Унарные	+, -, !, ~, ++, --, (тип),
Типа умножения (мультипликативные)	*, /, %
Типа сложения (аддитивные)	+, -
Сдвига	<<, >>
Отношения и проверки типа	<, >, is, …
Проверки на равенство	==, !=
Поразрядные логические	&, ^, |
Условные логические	&&, ||
Условная	?:
Присваивания	=, *=, /=,…

Операции и выражения

Приоритеты и ассоциативность операций языка

F (i) + G(i++) * H(i) (**)

(2 + 3) * 2

2 + 3 * 2

a + b + c означает (a + b) + c,
a = b = c означает a = (b = c) .

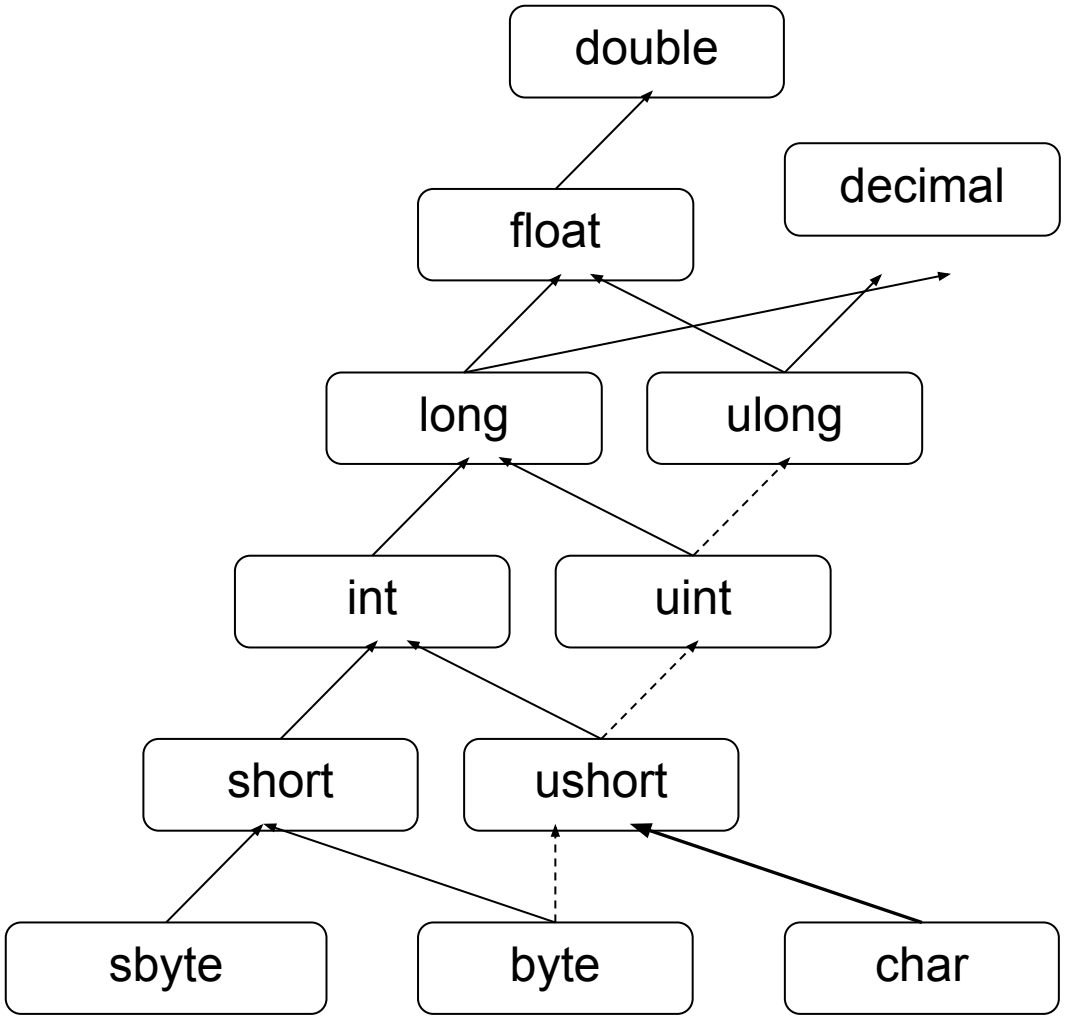
Тип результата выражения

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
```

Неявные арифметические преобразования типов в C#



при отсутствии линии возникает ошибка компиляции

Исключение

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
```

Операции отрицания - ! ~

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

a = 00000011

~a = 11111100

b = 11000001

~b = 00111110

c = 01111110

~c = 10000001

Явное преобразование типа

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

Операции сдвига (<< и >>)

Применяются к целочисленным операндам.

Сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом.

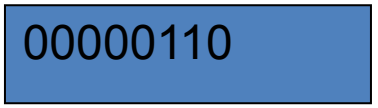
При *сдвиге влево* (<<) освободившиеся разряды обнуляются.

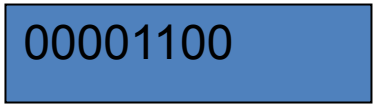
При *сдвиге вправо* (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае.

Стандартные операции сдвига определены для типов int, uint, long и ulong.

Операции сдвига (<< и >>)

```
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 && и ||

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

(a < b) || (a < c)
Если первое условие истинно, второе не вычисляется

Условная операция

операнд_1 ? операнд_2 : операнд_3

Операнд_1 — выражение, для которого существует неявное преобразование к логическому типу.
Если результат вычисления операнда_1 равен true, результат операции — значение **операнда_2**,
иначе — значение **операнда_3**.

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

Операция присваивания

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

Операция может использоваться в программе как законченный оператор.

переменная = выражение

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

Правый операнд операции присваивания должен иметь 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.

Консольный ввод-вывод

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

Результат работы программы:

3 y = 4,12

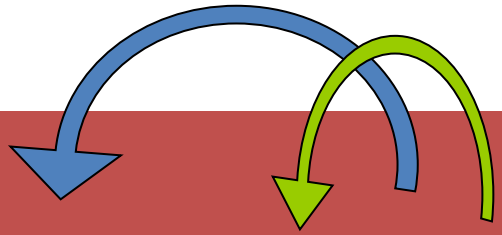
d = 600 s = Вася

Вывод на консоль

```
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 );

          }
     }
}
```



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.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

Формат место заполнителя

{ номер [, длина] [: формат] }

номер – номер элемента в списке вывода (может идти не по порядку и повторяться)

длина – количество позиций под значение. Если длина отрицательная, значение выравнивается по левому краю, иначе - по правому.

формат – строка формата для выводимого значения (описатели формата на следующем слайде)

Примеры:

{0,-6:F4}

{2:X4}

{1:hh}

Описатели формата

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

Операции отрицания - ! ~

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

Операции отрицания - ! ~

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

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.
        }
   }
}
```

Операции отрицания - ! ~

```
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() );
        }
   }
}
```

СПАСИБО ЗА ВНИМАНИЕ!

ИНЖЭКОН ENGECON

ФИНЭК
Санкт-Петербург

СПб ГУСЭ

ОБЪЕДИНЯЯ ЛУЧШЕЕ