*Керов Леонид Александрович*

# ВСТУПЛЕНИЕ К ЦИКЛУ СТАТЕЙ ПО ЯЗЫКУ C#

Серия статей является переработанной версией учебного пособия по языку программирования C#[1] . Необходимость его написания появилась шесть лет назад, когда студенты Санкт-Петербургского филиала государственного университета – Высшей Школы Экономики проходили компьютерное тестирование по информатике в рамках федерального Интернет-экзамена в сфере профессионального образования (ФЭПО), который направлен на проверку выполнения требований Государственных образовательных стандартов профессионального образования. Тогда успехи студентов по разделу «Алгоритмизация и программирование» были довольно скромными, и на кафедре бизнес-информатики было принято решение о включении соответствующего раздела в программу по информатике, в связи с чем и было написано учебное пособие.

Весной 2008 года наши студенты снова проходили тестирование в рамках ФЭПО, и большинство из них получили оценку «отлично». При этом среди языков программирования, из которых можно было выбирать при ответах на тесты, появился язык C#.

Таким образом, можно считать, что цель написания серии статей – подготовить студентов первого – второго курсов для успешного прохождения контрольных мероприятий типа ФЭПО. Поскольку в тестах ФЭПО ограничиваются уровнем консольных приложений, то и в статьях используется тот же уровень.

Вторая цель – дать студентам базу знаний «нулевого уровня», позволяющую переходить ко второй итерации разговора о C#, посвященной разработке содержательных приложений, например клиентских Windows-приложений с использованием технологии Windosw Forms или Windows Presentation Foundations, Web-приложений класса ASP и т. п.

Планируемая тематика серии статей «нулевого уровня» по C#:

1. Знакомство с языком C#.
2. Объекты и типы в языке C#.
3. Операторы управления, исключения и потоки в языке C#.
4. Массивы и коллекции в языке C#.
5. Методы, классы, ссылки и конструкторы в языке C#.
6. Методы объектно-ориентированного программирования на языке C#.

[1] *Керов Л.А.* Методы объектно-ориентированного программирования на C# 2005: Учебное пособие. СПб: Издательство «ЮТАС», 2007. 164 с.

# ЗНАКОМСТВО С ЯЗЫКОМ C#

**Аннотация**

Большинство книг по разработке приложений на языке программирования C# предполагает знакомство читателя с основными конструкциями этого языка, ведущего свою родословную от языков C, C++ и Java. Вместе с тем, опыт преподавания информационных технологий в университете показывает, что на младших курсах разговор об этом языке разумно начинать с краткого введения, не предполагающего никакой предварительной подготовки.

Данная статья является первой из серии статей, посвященных изложению «нулевого уровня» языка C#. Рассматриваются основные понятия платформы Microsoft .NET, для которой был специально разработан языка C#, создание простейшего приложения в среде Visual Studio 2008, объявление и использование переменных, арифметические операторы и операции консольного ввода-вывода.

**Ключевые слова:** язык C#, платформа Microsoft .NET, среда Visual Studio, объявление и использование переменных, арифметические операторы, операции консольного ввода-вывода.

## 1. ЯЗЫК C# И ПЛАТФОРМА MICROSOFT .NET

Язык C# разработан в компании Microsoft в начале 2000-х годов для создания приложений на платформе Microsoft .NET. Эта платформа включает следующие основные компоненты (см. рис. 1):

- ***Visual Studio*** – среду для разработки приложений, поддерживающую несколько языков программирования, основным из которых является язык C# (в данной работе будут использоваться современные версии – Visual Studio 2008, C# 2008).
- Каркас ***.NET Framework*** (современная версия 3.5).

Каркас .NET Framework является надстройкой над операционной системой (например Windows XP), автоматически устанавливается при инсталляции Visual Studio и содержит:

- Обширную библиотеку базовых классов ***BCL*** (Base Class Library), которая доступна всем языкам программирования для платформы Microsoft .NET. Эта библиотека обеспечивает выполнение операций ввода-вывода, операций доступа к данным, операций работы с Windows-формами, операций работы с Web-формами и др.
- Общеязыковую среду выполнения программ ***CLR*** (Common Language Runtime), которая обеспечивает загрузку и выполнение .NET-программ, а также выполнение ряда низкоуровневых задач, таких как управление памятью.

Реализация CLR выполнена на основе следующих спецификаций:

- Спецификации общей системы типов ***CTS*** (Common Type System), которая описывает все встроенные типы данных и программные конструкции, поддерживаемые средой выполнения CLR (см. рис. 2).
- Общей языковой спецификации ***CLS*** (Common Language Specification), которая может рассматриваться как подмножество CTS. Это подмножество должно быть реализовано каждым языком программирования для платформы .NET (конкретный язык программирования может не поддерживать всех возможностей CTS).

Процесс разработки и выполнения приложения на языке C# может быть представлен в виде следующей схемы (см. рис. 3). Программа на языке C# с помощью компилятора этого языка преобразуется в двоичный код, который называется *управляемым программным кодом* (managed code) и содержит не команды процессора конкретного компьютера, а команды абстрактного языка ***CIL*** (Common Intermediate Language – общий промежуточный язык). Управляемый программный код упаковывается в компоновочный блок, который называется *сбор-*

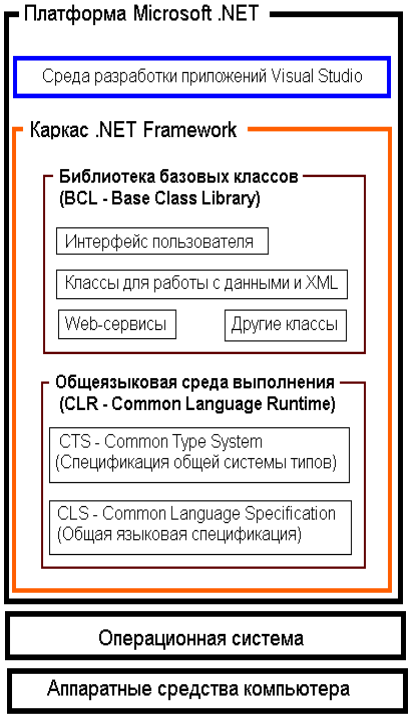


**Рис. 1.** Платформа Microsoft .NET

| Тип данных CTS | Ключевое слово C# | Размер в байтах значимого типа | Диапазон значений значимого типа |
|---|---|---|---|
| System.Byte | byte | 1 | 0 .. 255 |
| System.SByte | sbyte | 1 | -128 .. 127 |
| System.Int16 | short | 2 | -32,768 .. 32,767 |
| System.Int32 | int | 4 | -2,147,483,648 .. 2,147,483,647 |
| System.Int64 | long | 8 | -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 |
| System.Single | float | 4 | -3.402823e38 .. 3.402823e38 |
| System.Double | double | 8 | -1.79769313486232e308 .. 1.79769313486232e308 |
| System.Decimal | decimal | 16 | -79228162514264337593543950335 .. 79228162514264337593543950335 |
| System.Boolean | bool | 4 | True или False. |
| System.Char | char | 2 | Символ Unicode |
| System.String | string | Ссылочные типы | Строка символов Unicode |
| System.Object | object | | Объект |

**Рис. 2.** Основные встроенные типы данных CTS

*кой* (assembly) и сохраняется в исполняемом файле (*.exe) или в библиотечном файле (*.dll).

Кроме CIL-команд, сборка содержит:

• ***Метаданные***, которые подробно описывают особенности каждого типа внутри сборки (например базовый класс, реализуемые классом интерфейсы, а также полные описания всех членов класса).

• ***Манифест*** (manifest) – данные, описывающие саму сборку (текущая версия сборки, список ссылок на внешние сборки и др.).

Основной механизм среды выполнения программ CLR физически заключается в библиотеке **`mscoree.dll`** (Common Object Runtime Execution Engine – общий объектный модуль механизма выполнения). Когда поступает команда выполнить сборку, в память автоматически загружается модуль **`mscoree.dll`**, который, в свою очередь, загружает сборку в память. Если необходимо, из библиотеки базовых классов подгружаются нужные типы. Вся библиотека базовых классов разбита на ряд сборок, основной из которых является **`mscorlib.dll`** (Common Object Runtime Library).

После загрузки сборки механизм среды выполнения программ активизирует JIT-компилятор (Just In Time), который преобразует CIL-код в машинный код используемого процессора. Затем механизм среды выполнения запускает машинный код на выполнение. Таким образом, программа на языке C# в действительности выполняется в виде

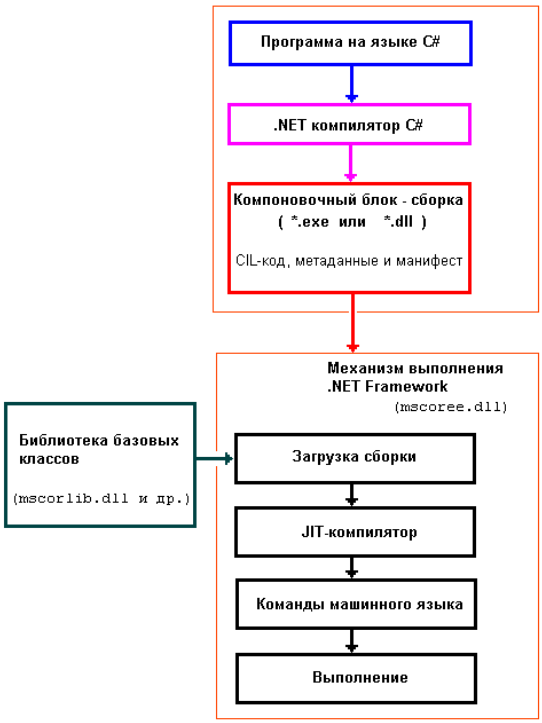


**Рис. 3.** Процесс разработки и выполнения приложения на языке C#

машинного кода, хотя первоначально она была скомпилирована в CIL-код. То есть программа будет выполняться так же быстро, как если бы она с самого начала была скомпилирована в машинный код. Использование 2-уровневой компиляции позволяет программисту создавать только один блок программного кода (сборку), который с помощью JIT-компиляции можно выполнять на компьютерах с разной архитектурой.

## 2. ПРОСТЕЙШЕЕ ПРИЛОЖЕНИЕ НА C#

Следуя традиции, рассмотрение языка C# начнем с простейшего приложения, которое выводит на экран текст «Hello, World!». Сначала запустим инструментальную систему Visual Studio 2008. Щелкнем по кнопке Пуск, затем из главного меню Windows выберем: **Программы → Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008.**

При работе с Visual Studio приложение создается в виде проекта (project), который представляет собой множество файлов, необходимых для создания приложения. Создадим пустой проект – пустое множество файлов, в которое затем будем добавлять необходимые файлы (см. рис. 4):

- Из меню **File** выберем пункт **New**, затем – команду **Project**.

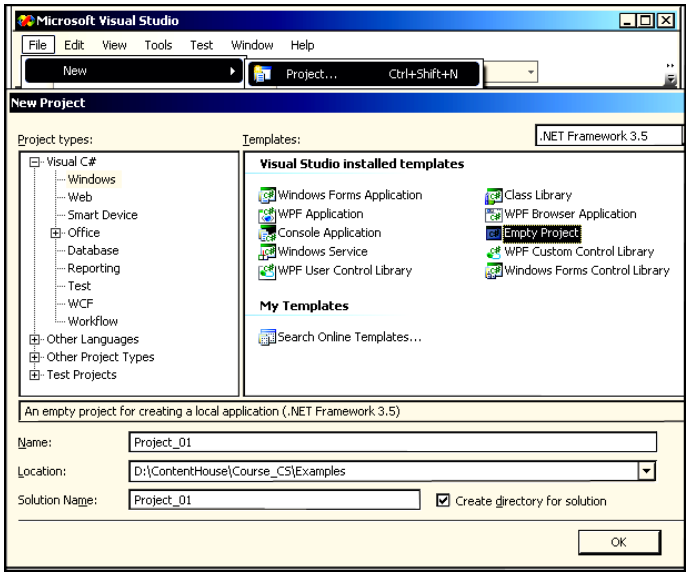


**Рис. 4.** Создание пустого проекта

- Появится диалоговое окно **New Project**. В области **Project types** раскроем пункт **Visual C#** и выберем элемент **Windows**.
- В области **Templates** выберем элемент **Empty Project**.
- В текстовое поле **Name** введем имя проекта – непустую последовательность символов, содержащую латинские буквы, цифры и знаки подчеркивания. Первым символом имени должна быть латинская буква, буквы верхнего и нижнего регистров считаются разными символами.
- В поле **Location** укажем путь к каталогу, в котором должна быть создана папка проекта.
- В заключение щелкнем кнопку «OK».

В результате выполненных действий будет создан пустой проект. В любой момент работы над проектом его текущее содержимое можно увидеть в окне **Solution Explorer**. Если этого окна нет на экране, то из меню **View** следует выбрать команду **Solution Explorer**. Мы создали пустой проект, поэтому в окне **Solution Explorer** пока нет никакой информации о файлах.

Для написания программы добавим в проект пустой файл кода:

- Из меню **Project** выберем команду **Add New Item**.
- В появившемся диалоговом окне выберем элемент **Code File**.
- Если необходимо, в текстовом поле **Name** отредактируем имя файла (файлы с кодом программы на языке C# имеют расширение **cs**).
- В заключение щелкнем кнопку «Add».

В проект будет добавлен пустой файл для написания программы, содержимое которого отображается в клиентской области окна. Используя клавиатуру, введем текст программы (листинг 1) на языке C# .

Первая строка программы начинается с двух символов слеш – это комментарий. Весь программный код на языке C# заключен внутри класса, определение кото-

рого начинается с ключевого слова **class** и который имеет имя **P01**. Фигурные скобки ограничивают содержимое класса.

**Листинг 1**

```
//Программа на языке C#
class P01
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, world");
    }
}
```

Класс содержит определение функции **Main()**, которая является точкой входа в программу на языке C#. Этот язык является чувствительным к регистру, поэтому отметим, что имя функции **Main()** должно начинаться с заглавной буквы.

После имени функции в круглых скобках перечисляются аргументы функции, а перед именем функции указывается тип её результата. Если функция не имеет аргументов (как в данном случае), то указываются пустые круглые скобки. Указание в качестве типа результата ключевого слова void означает, что функция не выдает никакого результата.

Функция **Main()** определена с использованием ключевых слов **public** и **static**:

- Ключевое слово **public** показывает, что к функции можно обращаться извне класса, в котором она определена
- Ключевое слово **static** показывает, что функцией класса можно пользоваться без создания объекта этого класса

Тело функции ограничено фигурными скобками и содержит оператор, который завершается точкой с запятой (;):

```
System.Console.WriteLine("Hello, world");
```

Этот оператор является вызовом статической функции **WriteLine()**, которая определена в классе **Console** пространства имен **System**. Аргументом функции является строка символов, которая выводится на экран.

В языке C# символьные строки заключаются в двойные кавычки и должны располагаться на одной строчке текста. Если текст длинный, то он разбивается на несколько строк символов, между которыми указывается знак плюс (+).

Код класса **Console** находится в сборке **mscorlib.dll**. Во время компиляции программы компилятор C# непосредственно обращается к файлу **mscorlib.dll** и получает из метаданных этого файла информацию о том, что в пространстве имен **System** находится класс **Console**, в котором определена функция **WriteLine()** с аргументом типа **string**. В итоге компилятор устанавливает в компоновочном блоке программы ссылку на файл **mscorlib.dll**. Когда программа выполняется, механизм среды выполнения программ (CLR) связывает программу с кодом функции **WriteLine()** в библиотеке **mscorlib.dll**.

Заметим, что компилятор C# обращается к библиотеке **mscorlib.dll** по умолчанию. Однако имеется много других DLL-библиотек, на которые в проекте нужно указать явные ссылки, для того чтобы компилятор смог найти расположенные в них классы. Для указания такой ссылки из меню **Project** следует выбрать команду **Add Reference**, а затем в одноименном диалоговом окне щелчком мыши указать нужную библиотеку.

Приведенную выше программу можно переписать так (см. листинг 2).

Пространство имен **System** указано в директиве **using** в начале программы, а перед вызовом **Console.WriteLine()** оно опу-

**Листинг 2**

```
using System;
class P01
{
    public static void Main()
    {
        Console.WriteLine("Hello, world");
    }
}
```

щено. Эта директива сообщает компилятору C#, что если ему нужно найти определение функции **Console.WriteLine()**, то он должен добавить префикс **System** в начале и затем искать определение функции **System.Console.WriteLine()**.

Для построения компоновочного блока (сборки) приложения нужно из меню **Build** выбрать команду **Build Solution**. Если при написании программы были допущены, ошибки, то в нижней части окна появится их перечень. Следует исправить первую ошибку и повторить процедуру построения исполнимого файла.

Этот процесс повторяется до тех пор, пока не будет выведено сообщение **Build succeeded**. После этого можно запустить исполняемый файл на выполнение, выбрав из меню **Debug** команду **Start Without Debugging**. На экран будут выведено окно с результатами работы программы.

## 3. ПЕРЕМЕННЫЕ

*Переменная* – это именованная область оперативной памяти компьютера, в которую может быть записано определенное значение. Каждая переменная программы до начала ее использования должна быть определена; например:

```
int x;
```

В определении переменной указывается ее тип (например **int**) и имя (например **x**). Имя позволяет различать переменные в программе и представляет собой непустую последовательность символов, которая может содержать латинские буквы, цифры и знаки подчеркивания; первым символом имени должна быть латинская буква; буквы верхнего и нижнего регистров считаются разными символами. *Тип* определяет расположение и размер области памяти, которая выделяется (в результате обработки оператора определения переменной) для хранения значения переменной (например если указан тип **int**, то в стеке выделяется область размером 32 разряда для хранения целого числа). В конце оператора определения переменной ставится точка с запятой (;).

**Листинг 3**

```
using System;
class P03
{
    public static void Main()
    {
        int x;
        x = 2;
        int y = 3;
        int z = x + y;
        Console.WriteLine(z);
    }
}
```

Чтобы задать начальное значение переменной (это называется инициализацией переменной) или изменить значение переменной во время работы программы, используется *оператор присваивания*, например:

```
x = 2;
```

Оператор присваивания изображается символом равенства. Слева от него указывается переменная, а справа – выражение (то есть константа, переменная или конструкция, построенная из констант или переменных с помощью операций и функций). Выполнение оператора присваивания начинается с вычисления значения выражения. Затем вычисленное значение помещается в область памяти, отведенную для значения переменной.

Определение переменной можно совместить с ее инициализацией, например:

```
int y = 3;
```

Для вывода значения переменной на экран монитора можно использовать функцию **WriteLine()**, например листинг 3.

## 4. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Для выполнения действий над числами в языке C# предусмотрены *арифметические операторы* (см. табл. 1).

Приведенная в листинге 4 программа иллюстрирует использование арифметических операторов.

**Листинг 4**

```
using System;
class P04
{   public static void Main()
    {   int x = 1, y = 2, z1 = x / y, z2 = x % y;
        double z3 = x / 2.0;
        Console.WriteLine(z1 + "\n" + z2 + "\n" + z3 + "\n" +
            (x++) + "\n" + (++y) + "\n" + x + "\n" + y);
    }
}
```

Первый оператор в теле функции **Main()** показывает, что в одном операторе можно определить несколько переменных одного и того же типа. При этом переменные перечисляются через запятую, каждая переменная (или только некоторые из них) могут быть инициализированы.

Отметим, что операция деления выполняется по-разному для целых и дробных чисел. Результатом деления одного целого числа на другое целое число является целая часть от деления; остаток отбрасывается (например, значение выражения 1/2 равно 0). В результате деления одного дробного числа на другое получается дробное число. Если делимое является целым числом, а делитель – дробным числом (или наоборот), то целое число преобразуется в дробное число с нулевой дробной частью, и выполняется деления одного дробного числа на другое.

Если оператор сложения применяется к строке символов, то он означает операцию конкатенации (сцепления) строк. Если

**Табл. 1**

| Оператор | Семантика |
|---|---|
| + | сложение |
| - | вычитание |
| * | умножение |
| / | деление |
| % | остаток от деления |
| ++ | инкремент |
| -- | декремент |

же одно слагаемое является числом, а второе – строкой символов, то число преобразуется в строку символов, и выполняется операция конкатенации строк. Для преобразования числового значения в строку символов в языке C# предусмотрена функция **ToString()**: указывается подлежащий преобразованию объект, оператор «точка» и эта функция. В предыдущем примере функция **ToString()** добавляется компилятором автоматически. При написании программы ее можно указать явно (см. листинг 5).

**Листинг 5**

```
using System;
class P04
{
    public static void Main()
    {
        int x = 1, y = 2, z1 = x / y, z2 = x % y;
        double z3 = x / 2.0;
        Console.WriteLine(z1.ToString() + "\n" + z2.ToString()
            + "\n" + z3.ToString() + "\n" + (x++).ToString() +
            "\n" + (++y).ToString() + "\n" + x.ToString() + "\n"
            + y.ToString());
    }
}
```

Оператор инкремента увеличивает значение операнда на единицу

```
x++   эквивалентно   x = x + 1
++x   эквивалентно   x = x + 1
```

Оператор декремента уменьшает значение операнда на единицу:

```
x--   эквивалентно   x = x - 1
--x   эквивалентно   x = x - 1
```

Если оператор инкремента или декремента применяется к отдельной переменной, то его префиксная и постфиксная формы имеют один и тот же смысл: переменная увеличивается или уменьшается на единицу. Если же оператор инкремента или декремента применяется к переменной, которая является частью некоторого выражения, то:

– если оператор записан в *префиксной* форме, *сначала* переменная увеличивается (или уменьшается) на единицу, а затем это новое значение переменной используется для вычисления выражения;

– если оператор записан в *постфиксной* форме, для вычисления выражения используется **старое значение переменной**, а затем переменная увеличивается (или уменьшается) на единицу.

Нередко оператор присваивания используется для увеличения значения некоторой переменной, уменьшения значения переменной и т. п. В этом случае удобно использовать составной оператор присваивания:

```
x = x + y;   эквивалентно   x += y;
x = x - y;   эквивалентно   x -= y;
x = x * y;   эквивалентно   x *= y;
x = x / y;   эквивалентно   x /= y;
x = x % y;   эквивалентно   x %= y;
```

## 5. ФОРМАТИРУЕМЫЙ ВЫВОД ДАННЫХ

В предыдущих примерах функция `WriteLine()` имела один аргумент, значение которого выводилось на экран монитора. Функция `WriteLine()` может иметь несколько аргументов (см. листинг 6).

Первый аргумент является строкой, которая формируется и выводится на экран. В этой строке могут быть указаны обычные символы, управляющие символы и числа в фигурных скобках. Обычные символы просто выводятся на экран. Управляющие символы имеют специальный смысл, например: `\t` – перевод курсора к ближайшему символу табуляции, `\n` – перевод курсора на новую строку. Отличительным признаком управляющего символа является указанный перед ним символ *backslash* (\). Полученная таким образом последовательность из двух символов называется *escape*-последовательностью.

Числа в фигурных скобках называются маркерами. *Маркер* – это номер выражения, которое указывается после первого аргумента функции `WriteLine()`. Нумерация начинается с нуля, то есть первое выражение имеет номер 0, второе – номер 1 и т.д. При выполнении форматированного вывода вместо маркеров подставляются значения выражений, указанных после строки форматирования: вместо маркера с номером 0 подставляется значение первого выражения, вместо маркера с номером 1 – значение второго выражения и т. д. Сформированный таким образом первый аргумент функции `WriteLine()` выводится на экран монитора.

**Листинг 6**

```
using System;
class P06
{
    public static void Main()
    {
        double x = 3.14, y = 4;
        Console.WriteLine(" x = {0}     \t y = {1}",    x, y);
        Console.WriteLine(" x = {0:C}   \t y = {1:P}",  x, y);
        Console.WriteLine(" x = {0:F1} \t y = {1:F2}", x, y);
    }
}
```

**Листинг 7**

```
using System;
class P07
{
    public static void Main()
    {
        Console.Write("Ваше имя            : ");
        string name = Console.ReadLine();
        Console.Write("Номер Вашей группы : ");
        int group = int.Parse(Console.ReadLine());
        Console.WriteLine("\nЗа компьютером работает " +
        "{0} из группы {1}", name, group);
    }
}
```

После маркера в фигурных скобках можно указать двоеточие и формат вывода, например:

- **C** – значение числа рассматривается как количество денег (Currency); оно выводится на экран с указанием денежной единицы, которая задается при установке операционной системы Windows.
- **P** – значение числа рассматривается как количество процентов (*Percent*); оно умножается на 100 и выводится на экран с указанием символа процента.
- **Fn** – значение числа выводится на экран с указанием *n* цифр в дробной части числа (*Fixed*).

## 6. ВВОД ДАННЫХ

Класс **Console** содержит функцию **ReadLine()**, которая не имеет аргументов и позволяет ввести с клавиатуры некоторое значение в программу:

- Значением функции **ReadLine()** является текст, символы которого набирались на клавиатуре
- Введенный текст можно присвоить в качестве значения переменной (например переменной **name**), которая имеет тип **string**.

Для ввода значений других типов (например **int**) необходимо преобразовывать введенную последовательность символов в объект соответствующего типа с помощью функции **Parse()**:

- Аргументом функции **Parse()** является значение функции **ReadLine()**.
- Перед функцией **Parse()** через точку указывается тип (например **int**), к которому нужно преобразовать введенную последовательность символов.
- Если преобразование невозможно, то программа выведет диагностическое сообщение об ошибке и остановится.

Если строка форматирования не помещается в одной строчке программы, то ее можно разбить на несколько строк, соединив их знаком «плюс» (то есть оператором конкатенации) (листинг 7).

## 7. ПРИСВОЕНИЕ ПЕРЕМЕННОЙ СЛУЧАЙНОГО ЗНАЧЕНИЯ

При написании программ (например игровых) возникает потребность использовать числа, которые порождаются случайным образом. Для этой цели в пространстве имен **System** предусмотрен класс **Random**, содержащий функцию **Next()** (см. табл. 2).

**Табл. 2**

| Тип результата | Функция | Семантика |
|---|---|---|
| **int** | **Next(int a, int b)** | Возвращает случайное целое число из диапазона **[a, b-1]** |

Для использования функции **Next()** предварительно нужно определить переменную типа **Random** (например **x**), которая должна быть проинициализирована выражением **new Random()**. Переменная типа **Random** указывается через точку перед функцией **Next()**. В приведенной в листинге 8 программе моделируется бросание игральной кости.

**Листинг 8**

```
using System;
class P08
{   public static void Main()
    {   Random x = new Random();
        int y1 = x.Next(1, 7);
        int y2 = x.Next(1, 7);
        Console.WriteLine(
            " Бросок 1 : {0}\n Бросок 2 : {1}", y1, y2);
    }
}
```

## Литература

1. *Керов Л.А.* Методы объектно-ориентированного программирования на C# 2005: Учебное пособие. СПб: Издательство «ЮТАС», 2007. 164 с.

2. *Нэш Т.* C# 2008: ускоренный курс для профессионалов: Пер. с англ. М.: ООО «И.Д. Вильямс», 2008. 576 с.

3. *Павловская Т.А.* C#. Программирование на языке высокого уровня. Учебник для вузов. СПб: Питер, 2007. 432 с.

4. *Троелсен Э.* Язык программирования C# 2005 и платформа .NET 2.0. 3-е издание.: Пер. с англ. М.: ООО «И.Д. Вильямс», 2007. 1168 с.

5. *Шилдт Г.* C#: учебный курс. СПб: Питер; К.: Издательская группа BHV, 2003. 512 с.

**Abstract**

The majority of books devoted to the applications development in the programming language C# assumes an acquaintance of the reader with the basic statements of this language leading the family tree from languages C, C ++ and Java. At the same time, experience of teaching of computer science in university demonstrates, that usage of this language is reasonable to begin with some brief introduction which are not assume any preliminary knowledge.

Given article is first of a series of articles, devoted to «a zero level» of language C#. The basic concepts of platform Microsoft .NET are considered. Language C# has been specially developed for this platform. Creation of the elementary application in Visual Studio 2008 is considered. The declaration and use of variables, use of arithmetic operators, use of input-output operators are considered.



***Керов Леонид Александрович, кандидат технических наук, старший научный сотрудник, доцент, заведующий кафедрой бизнес-информатики Санкт-Петербургского филиала государственного университета – Высшей Школы Экономики при Правительстве РФ,***

***kerov@hse.spb.ru***