Керов Леонид Александрович

ТИПЫ И ЭКЗЕМПЛЯРЫ ТИПОВ В ЯЗЫКЕ C#

Аннотация

Данная статья является второй из серии статей, посвященных изложению «нулевого уровня» языка C#. Рассматриваются понятия типа и экземпляра типа, встроенные типы языка C#, определение новых типов с помощью перечислений и структур, неявное и явное приведение типов.

Ключевые слова: язык C#, экземплярытипов, встроенные типы.

1. ПОНЯТИЕ ТИПА И ЭКЗЕМПЛЯРА ТИПА

Тип – это определенное множество данных и функций, которые в совокупности образуют информационную модель некоторого класса объектов (см. например рис. 1).

- Тип **`DateTime`** содержит определения данных для представления некоторой временной точки в григорианском календаре и определения ряда функций для работы с такими данными. Данные этого типа включают номер дня (**`day`**), месяца (**`month`**), года (**`year`**), часа (**`hour`**), минуты (**`minute`**), секунды (**`second`**). Примером функции может служить **`AddDays()`** для добавления некоторого количества дней к указанной дате с целью получения новой даты
- Тип **`int`** содержит определение данных для представления значения целого числа и определение операций для действий над целыми числами: сложение (**`+`**), вычитание (**`-`**), умножение (**`*`**) и т. д.
- Тип **`Math`** содержит определения математических функций, таких как извлечение квадратного корня **`Sqrt()`**, вычисление синуса **`Sin()`** и т. д.

Если тип содержит определения данных и функций, то можно создавать объекты этого типа. В качестве примера приведем программу, в которой создается и используется объект **`birth`** типа **`DateTime`** с данными **`1948.12.07 18:00:00`** (см. листинг 1, рис. 2).

Созданные таким образом объекты называются *экземплярами* (**`instance`**) соответствующего им типа. Если тип не содержит

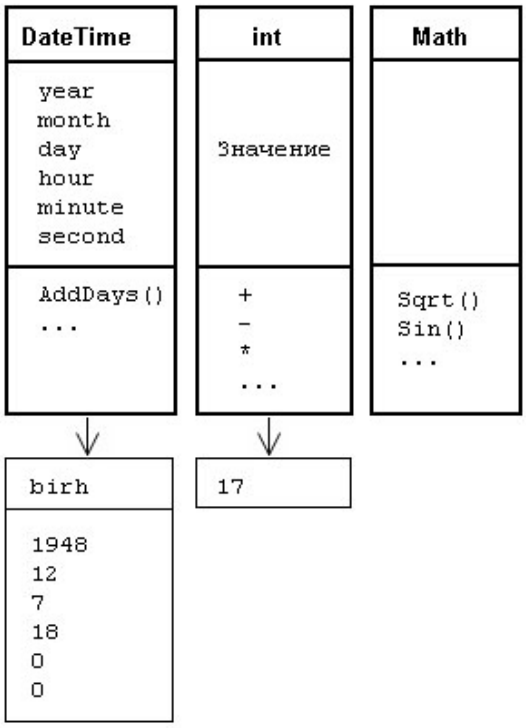

Рис. 1. Примеры типов в языке C#

© Л.А. Керов, 2009

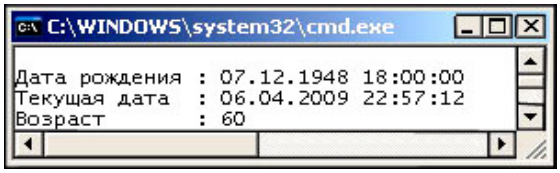

Рис. 2. Результат работы программы с объектом **birth**

определений данных, а содержит только определения функций, то мы не можем создавать объекты указанного типа и должны использовать только его функции. Примером может служить тип **Math**.

Объект **birth** типа **DateTime** имеет как имя, так и значения элементов данных. Объект типа **int** выглядят несколько иначе: у него нет имени, а только значение данных. Это является особенностью *встроенных типов*, соответствующих физическим типам самого компьютера. К ним, в частности, относятся числовые типы. Функции встроенного типа реализуются как встроенные операторы языка программирования, например: **+**, **-**, ***** и т. д.

Программа на языке C# состоит из типов, предназначенных для определения объектов (см. рис. 3). Вся логика программы может быть записана только внутри того или иного типа. Как следствие, в языке C# нет таких понятий, как глобальная переменная или глобальная функция. Типов в языке C# довольно много, и в целях упорядочивания они подразделяются на логически связанные совокупности, называемые *пространствами имен* (**namespace**).

2. ВСТРОЕННЫЕ ТИПЫ

Язык C# содержит восемь встроенных типов для работы с целыми числами. Эти типы различаются размерами ячейки памяти для хранения значений и, соответственно, диапазонами значений (см. табл. 1).

Каждый числовой тип содержит определения статических свойств MinValue и MaxValue, значениями которых являются, соответственно, минимальное и максимальное значение объекта данного типа (см. листинг 2, рис. 4).

Для работы с дробными числами язык C# содержит два типа с плавающей точкой (см. табл. 2).

Если в тексте программы содержится дробное число, например, **3.14**, то компи-

Табл. 1

Количество разрядов ячейки	Тип числа со знаком	Тип числа без знака
8	**sbyte**	**byte**
16	**short**	**ushort**
32	**int**	**uint**
64	**long**	**ulong**

Табл. 2

Тип с плавающей точкой	Порядок	Мантисса	Всего бит
float	8	24	32
double	11	53	64

Листинг 1

```
using System;
class P01
{
    public static void Main()
    {
        DateTime birth = new DateTime(1948, 12, 7, 18, 0, 0);
        DateTime now = DateTime.Now;
        TimeSpan age = now - birth;
        int days = age.Days;
        Console.WriteLine("\nДата рождения : {0}" +
                          "\nТекущая дата  : {1}" +
                          "\nВозраст       : {2}",
                          birth, now, days / 365);
    }
}
```

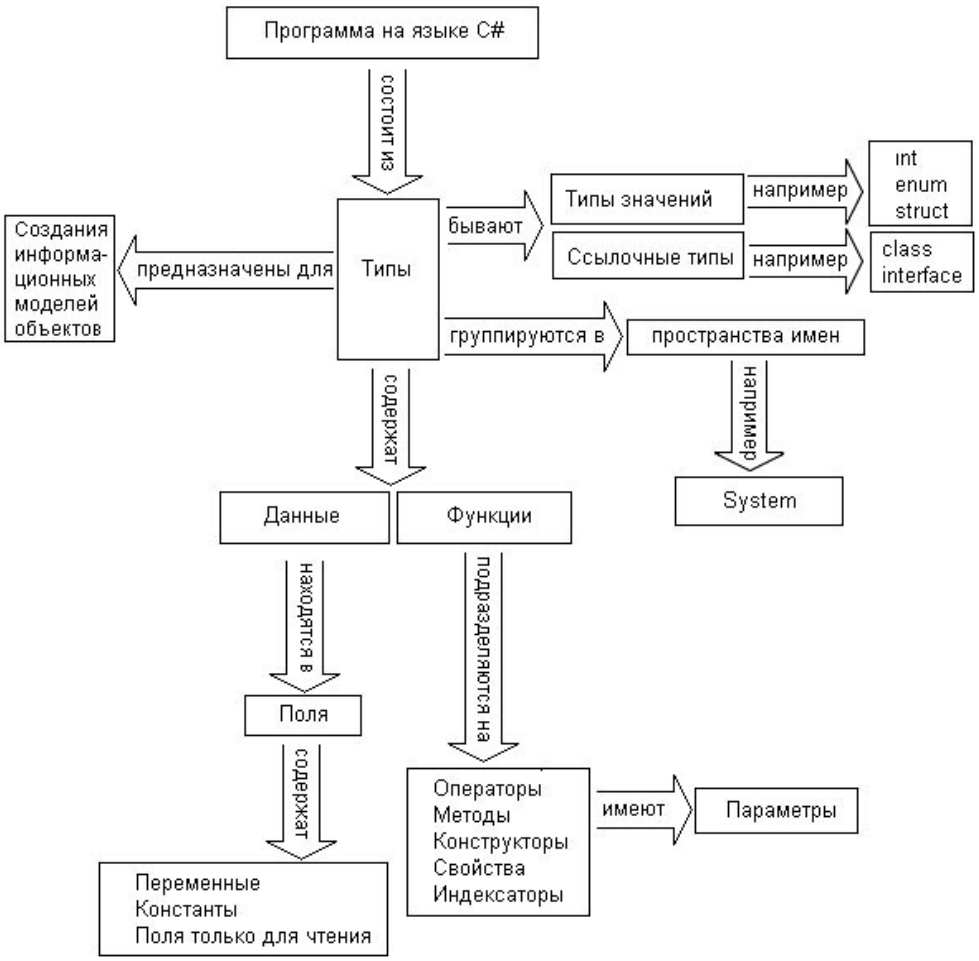

Рис. 3. Программа на языке C# состоит из типов

Листинг 2

```
using System;
class P02
{
    public static void Main()
    {   Console.WriteLine("sbyte \t [ {0}, {1} ]",
            sbyte.MinValue, sbyte.MaxValue);
        Console.WriteLine("byte \t [ {0}, {1} ]",
            byte.MinValue, byte.MaxValue);
        Console.WriteLine("short \t [ {0}, {1} ]",
            short.MinValue, short.MaxValue);
        Console.WriteLine("ushort \t [ {0}, {1} ]",
            ushort.MinValue, ushort.MaxValue);
        Console.WriteLine("int \t [ {0}, {1} ]",
            int.MinValue, int.MaxValue);
        Console.WriteLine("uint \t [ {0}, {1} ]",
            uint.MinValue, uint.MaxValue);
        Console.WriteLine("long \t [ {0}, {1} ]",
            long.MinValue, long.MaxValue);
        Console.WriteLine("ulong \t [ {0}, {1} ]",
            ulong.MinValue, ulong.MaxValue);
    }
}
```

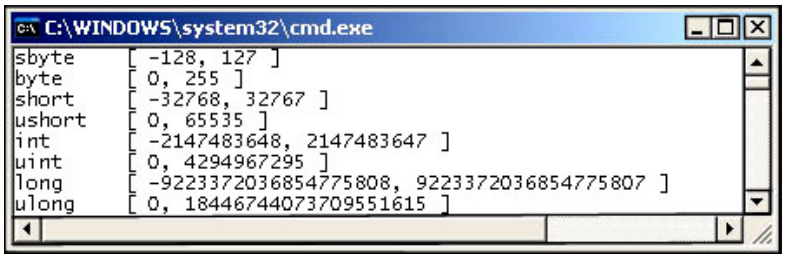

Рис. 4. Диапазоны значений целочисленных типов

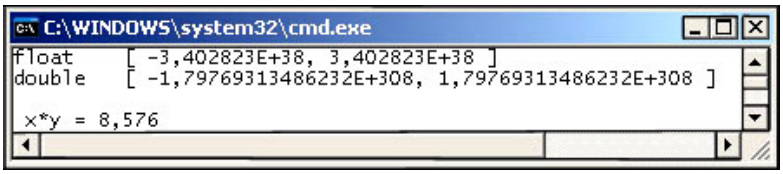

Рис. 5. Диапазоны значений
и форматированный вывод дробных типов

лятор рассматривает его как имеющее тип **double**. Чтобы указать, что **3.14** имеет тип **float**, нужно добавить суффикс **f**: **3.14f**. Для вывода на экран числа с плавающей точкой предусмотрен формат вывода **Fn**: значение числа выводится на экран с указанием **n** цифр в дробной части числа (см. листинг 3, рис. 5).

Для выполнения финансовых расчетов в языке C# предусмотрен тип **decimal**, который содержит 128 двоичных разрядов для хранения данных.

Листинг 3

```
using System;
class P03
{   public static void Main()
    {   Console.WriteLine("float \t [ {0}, {1} ]",
            float.MinValue, float.MaxValue);
        Console.WriteLine("double \t [ {0}, {1} ]",
            double.MinValue, double.MaxValue);
        double x = 2.73; float y = 3.1415f;
        Console.WriteLine("\n x*y = {0:F3}v, x * y);
    }
}
```

Листинг 4

```
using System;
class P04
{
    public static void Main()
    {
        Console.WriteLine("\ndecimal\t [ {0}, {1} ]",
            decimal.MinValue, decimal.MaxValue);
        //Пример расчета баланса после начисления процентов
        decimal balance = 1000.00M;
        decimal rate = 0.1M;
        Console.WriteLine(
            "\nСумма на счете  : {0:C}\n" +
              "Процент прибыли : {1:P}\n» +
              "Сумма после начисления процентов : {2:C}",
              balance, rate, balance + balance * rate);
    }
}
```

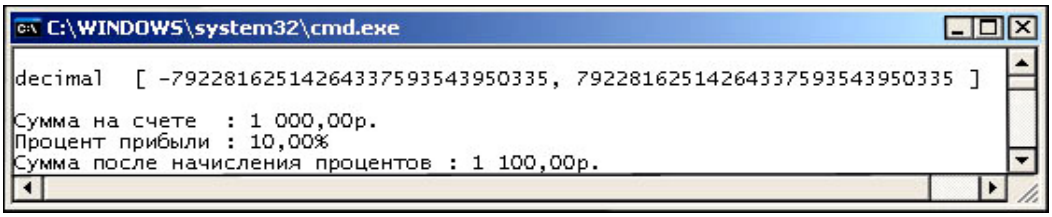

Рис. 6. Диапазон значений и пример использования типа **decimal**

Чтобы указать, что, например, число **3.14** имеет тип **decimal**, нужно добавить суффикс **M** или **m**, например: **3.14M** (см. листинг 4, рис. 6).

C# содержит тип **bool**, который имеет два значения **true** и **false**. Значения типа **bool** имеют результаты операций сравнения (табл. 3).

Для выполнения действия над объектами типа **bool** в языке C# определены следующие логические операторы (табл. 5). Смысл логических операторов может быть определен с помощью таблицы истинности (табл. 4).

Различие между сокращенной и обычной операцией «И» демонстрирует следующая программа (см. листинг 5, рис. 7).

Встроенный тип **char** используется для представления отдельных символов. Этот тип использует 16-битную кодировку *Unicode*, а не 8-битовую кодировку *ASCII* (как, например, в языке C++). Набор символов *ASCII* составляет подмножество *Unicode* с кодами в диапазоне [0,127]. Значения типа **char** заключаются в апострофы, например: **char ch = 'x'**;

Встроенный тип **string** используется для представления строк символов. Значения этого типа заключаются в двойные кавычки. Переменной типа **char** можно присвоить символ строки, указав его порядковый номер в квадратных скобках после имени переменной типа **string** (нумерация начинается с нуля). Число символов в переменной типа **string** можно определить с помощью свойства **Length** (см. листинг 6, рис. 8).

Табл. 3

Обозначение операции	Смысл операции
==	равно
!=	не равно
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

Табл. 4

x	y	x & y	x | y	x ^ y	!x
false	false	false	false	false	true
true	false	false	true	true	false
false	true	false	true	true	
true	true	true	true	false	

Табл. 5

Обозначение операции	Смысл операции
&	И (Вычисляются оба операнда)
|	ИЛИ (Вычисляются оба операнда)
^	исключающее ИЛИ
!	НЕ
&&	сокращенное И (Если 1-й операнд **false**, 2-й не вычисляется)
||	сокращенное ИЛИ (Если 1-й операнд **true**, 2-й не вычисляется)

Листинг 5

```
using System;
class P05
{
     public static void Main()
     {
          int x = 10, y = 0;
          Console.WriteLine((y != 0) && (x % y == 0));
          Console.WriteLine((y != 0) & (x % y == 0));
     }
}
```

Рис. 7. Различие между сокращенной и обычной операцией «И»

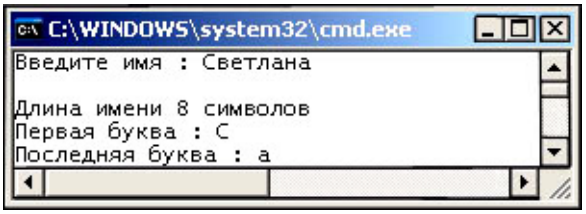

Рис. 8. Пример обработки символьной информации

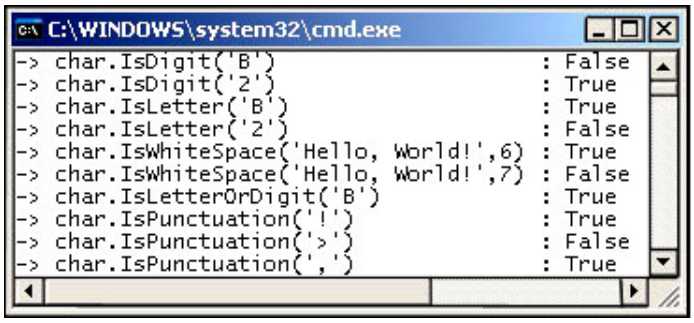

Рис. 9. Примеры лексического анализа символьной информации

Встроенный тип **System.Char**, синонимом для которого является слово **char**, содержит статические методы, позволяющие определить, является ли данный символ цифрой, буквой, пробелом или знаком пунктуации (см. листинг 7, рис. 9).

Синонимом встроенного типа **System.String** является слово **string**. Этот тип содержит, среди прочих, следующие члены (табл. 6).

Операции равенства и неравенства (**==** и **!=**) сравнивают содержимое двух строк. Для конкатенации строк можно использовать операцию «плюс» (**+**) или

Листинг 6

```
using System;
class P06
{    public static void Main()
     {   Console.Write("Введите имя : ");
          string x = Console.ReadLine(); char y = x[0];
          Console.WriteLine("\nДлина имени {0} символов" +
          "\nПервая буква : {1}\nПоследняя буква : {2}",
          x.Length, y, x[x.Length - 1]);
     }
}
```

Листинг 7

```
using System;
class P07
{
  public static void Main()
  {
    Console.WriteLine("-> char.IsDigit('B')                    : {0}",
        char.IsDigit('B'));
    Console.WriteLine("-> char.IsDigit('2')                    : {0}",
        char.IsDigit('2'));
    Console.WriteLine("-> char.IsLetter('B')                   : {0}",
        char.IsLetter('B'));
    Console.WriteLine("-> char.IsLetter('2')                   : {0}",
        char.IsLetter('2'));
     Console.WriteLine("-> char.IsWhiteSpace('Hello, World!',6) : {0}",
        char.IsWhiteSpace("Hello, World!", 6));
     Console.WriteLine("-> char.IsWhiteSpace('Hello, World!',7) : {0}",
        char.IsWhiteSpace("Hello, World!", 7));
    Console.WriteLine("-> char.IsLetterOrDigit('B')            : {0}",
        char.IsLetterOrDigit('B'));
    Console.WriteLine("-> char.IsPunctuation('!')              : {0}",
        char.IsPunctuation('!'));
    Console.WriteLine("-> char.IsPunctuation('>')              : {0}",
        char.IsPunctuation('>'));
    Console.WriteLine(v-> char.IsPunctuation(',')              : {0}",
        char.IsPunctuation(','));

  }
}
```

статический метод **String.Concat()** (см. листинг 8, рис. 10).

Строковые литералы могут содержать управляющие последовательности (табл. 7).

Перед строкой, которую требуется воспроизвести буквально, следует указать префикс «@» (см. листинг 9, рис. 11).

Объекты типа **string** обладают следующей особенностью: значение строки после ее создания изменить нельзя. Строки в языке C# неизменяемы. Если, например, строковой переменной присваивается новое значение, то фактически создается новый строковый объект, а исходная строка объяв-

Табл. 6

Член типа **System.String**	Описание
Length()	Свойство, возвращающее длину текущей строки
Contains()	Метод, проверяющий, содержит ли строковый объект указанную строку
Format()	Статический метод, имеющий такие же аргументы, как метод форматируемого вывода на экран **Console.WriteLine()**. Вместо вывода на экран строка возвращается как результат метода **Format()**
Replace()	Получение копии текущей строки, в которой произведена замена символа
Insert()	Получение копии текущей строки, дополненной указанными строковыми данными

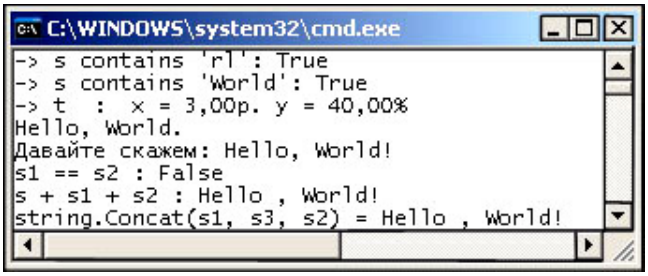

Рис. 10. Примеры обработки объектов типа **System.String**

ляется мусором и будут удалена сборщиком мусора.

Чтобы уменьшить число копирований строк, в пространстве имен **System.Text** определен тип **StringBuilder**. В отличие от типа **System.String**, этот тип обеспечивает прямой доступ к буферу строки. При создании объекта типа **StringBuilder** можно указать (через параметр конструктора) размер буфера, в котором будет размещаться объект. По умолчанию, этот размер равен 16. С помощью метода **Append()** можно добавить символы в объект типа **StringBuilder**, при этом размер буфера будет динамически увеличиваться (см. листинг 10, рис. 12).

3. ОПЕРЕДЕЛЯЕМЫЕ ТИПЫ: ПЕРЕЧИСЛЕНИЯ И СТРУКТУРЫ

При написании программы на языке C# можно объявлять свои собственные типы.

Табл. 7

Управляющая последовательность	Описание
\'	Вставляет в строковый литерал символ апострофа
\"	Вставляет в строковый литерал символ двойной кавычки
\\	Вставляет в строковый литерал символ обратной косой черты
\n	Вставляет в строковый литерал символ перехода на новую строку
\t	Вставляет в строковый литерал символ горизонтальной табуляции

Листинг 8

```
using System;
class P08
{ public static void Main()
  { string s = "Hello, World!";
     Console.WriteLine("-> s contains 'rl': {0}",
     s.Contains("rl"));
     Console.WriteLine("-> s contains 'World': {0}",
     s.Contains("World"));
     double x = 3, y = 0.4;
     string t = String.Format(" x = {0:C} y = {1:P}", x, y);
     Console.WriteLine("-> t  : {0}", t);
     Console.WriteLine(s.Replace('!', '.'));
     Console.WriteLine(s.Insert(0, "Давайте скажем: "));
     string s1 = "Hello ", s2 = "World!", s3 = ", ";
     Console.WriteLine("s1 == s2 : {0}", s1 == s2);
     string s4 = s1 + s3 + s2;
     Console.WriteLine("s + s1 + s2 : {0}", s4);
     Console.WriteLine("string.Concat(s1, s3, s2) = {0}",
     string.Concat(s1, s3, s2));
    }
}
```

Листинг 9

```
using System;
class P09
{ public static void Main()
  { string s3 = "Hello\tThere\tAgain";
    Console.WriteLine(s3);
    Console.WriteLine("Everyone loves \"Hello World\"");
     Console.WriteLine("C:\\MyApp\\bin\\debug");
    Console.WriteLine("All finished.\n");
    string finalString = @"\n\tString file: 'C:\MyApp\Strings'";
    Console.WriteLine(finalString);
    string myLongString = @"This is a very
        very
        very
        long string";
    Console.WriteLine(myLongString);
  }
}
```

Наиболее простыми механизмами для этого являются перечисления и структуры.

Перечисления предназначены для создания переменных, которые могут принимать значения из заранее заданного множества целочисленных констант. Это позволяет программисту не запоминать значения констант, а просто использовать их имена. Перечисления всегда являются целыми типами данных, по умолчанию, они имеют тип **int**. Если не указано значение явно, то:

– первому элементу перечисления присваивается нулевое значение,

– следующим элементам значения присваиваются в порядке возрастания.

При использовании перечисления сначала указывается его имя, затем через точку – имя элемента (см. листинг 11, рис. 13).

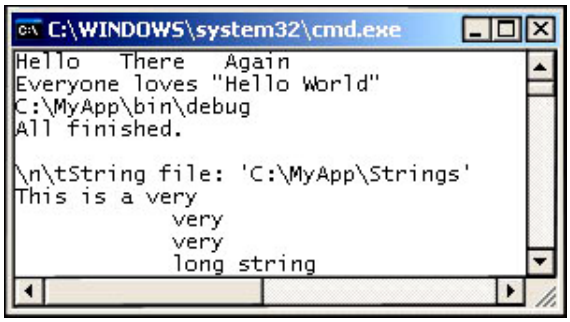

Рис. 11. Использование управляющих последовательностей и префикса «@»

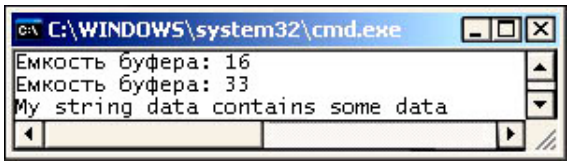

Рис. 12. Использование типа **StringBuilder**

Листинг 10

```
using System;
using System.Text;
class P10
{ public static void Main()
  { StringBuilder s = new StringBuilder("My string data");
    Console.WriteLine("Емкость буфера: {0}", s.Capacity);
    s.Append(" contains some data");
    Console.WriteLine("Емкость буфера: {0}", s.Capacity);
    Console.WriteLine(s);
  }
}
```

Листинг 11

```
using System;
class P11
{   public enum MyColor
    {   Red, Green = 30, Blue
    }
    public static void Main()
    {   Console.WriteLine(
        "\nЗначение цвета {0} \t равно {1}" +
        "\nЗначение цвета {2} \t равно {3}" +
        "\nЗначение цвета {4} \t равно {5}",
        MyColor.Red, (int)MyColor.Red,
        MyColor.Green, (int)MyColor.Green,
        MyColor.Blue, (int)MyColor.Blue);

        Console.WriteLine(
        "\nЗначение цвета {0} \t равно {1}" +
        "\nЗначение цвета {2} \t равно {3}" +
        "\nЗначение цвета {4} \t равно {5}",
        MyColor.Red,   MyColor.Red,
        MyColor.Green, MyColor.Green,
        MyColor.Blue,  MyColor.Blue);
    }
}
```

При написании программы на языке C# можно рассматривать некоторое множество переменных одного или различных типов в качестве компонентов более сложного типа. Построенный таким образом сложный тип называется *структурой*, а составляющие его компоненты называются *полями структуры*.

Объявив структуру, можно определять переменные этого сложного типа. Чтобы выделить память для значения такой переменной, используется ключевое слово **new**. Структуры можно создавать и без использования этого ключевого слова, но в этом случае необходимо выполнить инициализацию всех полей структуры до их использования. Если этого не сделать, компилятор выдаст сообщение об ошибке.

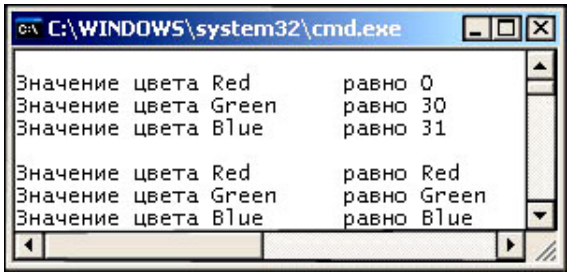

Рис. 13. Использование перечисления

Для обращения к полю структуры нужно указать имя структуры, затем символ «точка» (.) и имя поля структуры (см. листинг 12, рис. 14).

4. ПРИВЕДЕНИЕ ТИПОВ

Во всех языках программирования существует правило: любая операция (например, присваивание, сложение и т.д.) может применяться к данным одного и того же типа. Если в программе для какой-то операции указаны операнды разного типа, то перед выполнением операции они приводятся в одному типу. Если это невозможно, то такая ситуация рассматривается как ошибка.

В языке C# существует *неявное приведение типов*. Оно выполняется в тех случаях, когда приведение типа не приводит к потере данных. Например, значение типа **int** можно присвоить переменной типа **double**. При этом компилятор автоматически преобразует целое число в дробное число с нулевой дробной частью. Понятно, что никакие данные при этом не будут потеряны (см. листинг 13, рис. 15).

Листинг 12

```
using System;
class P12
{    public struct Student
     {    public string firstName;
          public string lastName;
          public int groupNumber;
     }
     public static void Main()
     {    Student std;
          //Student std = new Student();
          Console.Write("Введите имя          : ");
          std.firstName  =  Console.ReadLine();
          Console.Write("Введите фамилию      : ");
          std.lastName  =  Console.ReadLine();
          Console.Write("Введите номер группы: ");
          std.groupNumber  =  int.Parse(Console.ReadLine());
          Console.WriteLine();
          Console.WriteLine("За компьютером работает " +
          "{0} {1} из группы {2}",
          std.firstName, std.lastName, std.groupNumber);
     }
}
```

Если делается попытка, например, переменной типа **double** присвоить значение переменной типа **int**, то компилятор выдаст сообщение об ошибке, так как при этом возможна потеря данных (см. листинг 14, рис. 17).

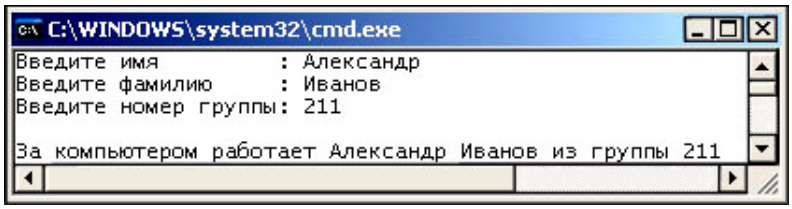

Рис. 14. Использование структуры

В таком случае следует использовать явное приведение типа, указав перед операндом в круглых скобках тот тип, которому следует выполнить приведение. При этом возможна потеря части данных, однако в программе явным образом разрешается это сделать (см. листинг 15, рис. 16).

Листинг 13

```
using System;
class P13
{
     public static void Main()
     {    int i = 7; double x;
          x = i;  //x = 7.0
          Console.WriteLine("x = {0:F3}", x);
     }
}
```

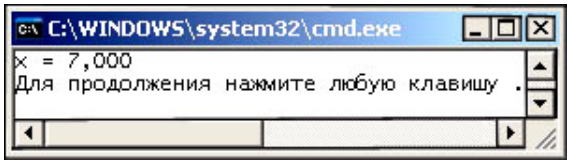

Рис. 15. Пример неявного приведения типов

5. ВЫЧИСЛЕНИЕ ЗНАЧЕНИЯ ВЫРАЖЕНИЯ

Выражение – это правило вычисления значения, которое имеет вид константы, переменной, или комбинации из констант,

Листинг 14

```
using System;
class P14
{   public static void Main()
    {   double x = 3.14;
        int i;
        i = x;
        Console.WriteLine("i = {0}", i);
    }
}
```

Листинг 15

```
using System;
class P14
{   public static void Main()
    {   double x = 3.14;
        int i;
        i = (int)x;
        Console.WriteLine("i = {0}", i);
    }
}
```

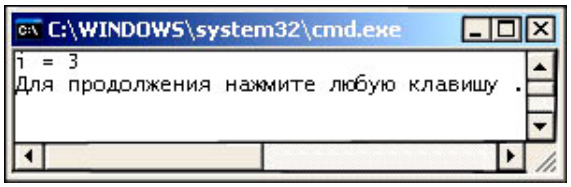

Рис. 16. Пример явного приведения типов

– если один операнд имеет тип **decimal**, то второй преобразуется к типу **decimal** (если при этом второй операнд имеет тип **float** или **double**, то это рассматривается как ошибка);

– операнд имеет тип **double**, то второй преобразуется к типу **double**;

– иначе если один операнд имеет тип **float**, то второй преобразуется к типу **float**;

– иначе если один операнд имеет тип **ulong**, то второй преобразуется к типу **ulong** (если при этом второй операнд имеет тип **sbyte**, **short**, **int** или **long**, то это рассматривается как ошибка);

– иначе если один операнд имеет тип long, то второй преобразуется к типу **long**;

– иначе если один операнд имеет тип uint, а второй операнд имеет тип **sbyte**, **short**, **int**, то оба преобразуется к типу **long**;

– иначе если один операнд имеет тип uint, то второй преобразуется к типу **uint**;

– иначе если ни одно из вышеуказанных правил не подошло, то оба операнда преобразуются к типу **int**.

Последнее правило называют «подтягиванием до **int**», иногда из-за него приходится использовать явное приведение типа, например: **byte b=10; b=(byte)b*b**.

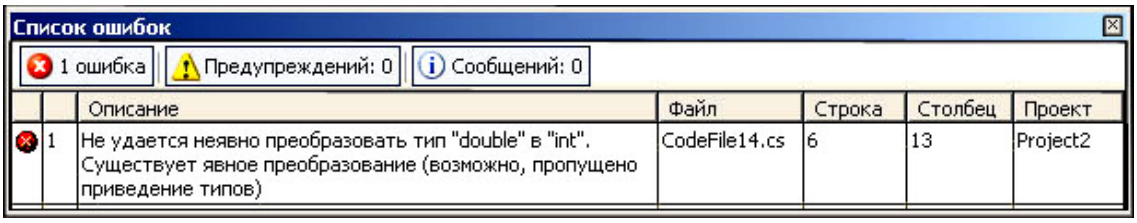

Рис. 17. Пример сообщения об ошибке приведения типа

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.

Тип операции	Операции в порядке убывания приоритета	Ассоциативность
Простая	**() [] . x++ x-- new typeof sizeof checked unchecked**	→
Унарная	**+ - ! ~ ++x --x (type)**	→
Мультипликативная	*** / %**	→
Аддитивная	**+ -**	→
Сдвиг	**<< >>**	→
Отношение	**< > <= >= is**	→
Равенство	**== !=**	→
Логическое И	**&** Вычисляются оба операнда	→
Логическое исключающее ИЛИ	**^** Вычисляются оба операнда	→
Логическое ИЛИ	**|** Вычисляются оба операнда	→
Условное И	**&&** Если 1-й операнд **false**, 2-й не вычисляется	→
Условное ИЛИ	**||** Если 1-й операнд **true**, 2-й не вычисляется	→
Условная	**?:** Пример: **x > y ? x : y;**	→
Присваивание	**= += -= *= /= %= <<= >>= |= &= ^=**	←

Рис. 18. Приоритет операций

Abstract

The article is the second one of the series of articles devoted to teaching «zero level» of the C# language. Concepts of type and of type instance, primitive data types, definition of new types by means of enumerations and structures, implicit and explicit cast 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

© Наши авторы, 2009.
Our authors, 2009.