Лабораторная работа №2

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И МЕХАТРОНИКИ

Кафедра информационных технологий и мехатроники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению лабораторных работ по дисциплине «Алгоритмизация и программирование»
для студентов специальности 6.050101 “Компьютерные науки”

Разработчик - доцент кафедры информационных технологий и мехатроники
кандидат технических наук, старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2015

Лабораторная работа №2
Исследование возможностей интегрированной среды разработки Visual C# для создания консольных приложений. Ввод-вывод данных в консольном приложении.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию консольных приложений ОС Windows с вводом и выводом данных.

1. Теоретические сведения

Любая программа работает с данными (числами, стро ками), которые могут вводиться пользователем или жестко прописываться в коде программы Эти данные хранятся либо на жестком диске (постоянные данные) либо в оперативной памя ти компьютера (временные данные). Под временными данными понимаются данные необходимые программе для рас четов. Так как процессор выполняет операции только над данными в оперативной памяти, то для работы с постоянными данными их необходимо загрузить в оперативную память. Для работы с данными, загруженными в память компьютера, в языке ассемблера используются адреса памяти, где хранятся данные, а в C# - имена. Имя, которое используется для работы, и есть переменная.

Переменная — это именованная область памяти, которой может быть присвоено определенное значение. Во время выполнения программы значение переменной может меняться. В C# все переменные должны быть объявлены до их использования. В объявлении переменной помимо ее имени необходимо указать, значения какого типа она может хранить, т.е. объявляется тип переменной. Основными типами данных являются:

- **byte** - для хранения 8-разрядных целых чисел без знака;
- **int** - для хранения 32-разрядных целых чисел со знаком;
- **float** - для хранения 32-разрядных чисел с плавающей точкой (вещественные числа обычной точности);
- **double** - для хранения 64-разрядных чисел с плавающей точкой (вещественные числа с двойной точностью);
- **char** - для хранения одного символа (16-битный, так как C# использует кодировку Unicode вместо 8-битной кодировки ASCII);
- **string** - для хранения строк из нескольких символов;
- **bool** - для хранения булевой переменной, которая принимает только одно значениеtrue или false.

Основным способом взаимодействия с пользователем в консольных приложениях, созданных Microsoft Visual Studio, является использование объекта Console. Методы, обеспечивающие отображение и ввод данных, пе речислены в табл.1.

Таблица 1. Методы объекта Console

Метод	Описание
Console.Read()	Обеспечивает ввод кода символа, набран ного на клавиатуре
Console.ReadLine()	Обеспечивает ввод строки сим волов, набран ной на клавиатуре. Для преобразования строки в число надо использовать, например, методы **Convert.ToInt32()** или **Convert.ToSingle()**
Console.Write(Str)	Выводит на экран (в окно консоли) строку **Str**, указанную в качестве параметра метода
Console.WriteLine(Str)	Выводит на экран (в окно консоли) строку **Str**, после чего переводит курсор в начало следующей строки

1.1. Базовый ввод-вывод с помощью класса Console

Для чтения из консоли существуют два метода: **Read()** и **ReadLine()**. Первый метод читает данные посимвольно, т. е. при обращении к методу он возвращает очередной символ из потока ввода, по умолчанию это окно консоли.

Все это выглядит следующим образом. Когда пользователь вводит строку в консоль и нажимает клавишу **<Enter>**, то метод возвращает код первого из введенных символов. Второй вызов метода вернет код второго символа. Обратите внимание, что метод возвращает числовой код, а не символ в виде типа данных **char**. Чтобы получить символ, нужно конвертировать число в тип данных **char**. Нижеприведенный фрагмент кода демонстрирует введение строки посимвольно.

Пример 1. В качестве примера создадим консольное приложение, которое выводит на экран результат ввода строки символов.

```
char c;
do {
    int x = Console.Read(); // чтение очередного символа
    c = Convert.ToChar(x); //  конвертирование в тип char
    Console.WriteLine(c); //  вывод символа в отдельной строке
   } while (c != ‘q');
```

Здесь запускается цикл **do … while**, который будет выполняться, пока пользователь не введет символ **q**. Внутри цикла первой строкой вызывается метод **Read()**, который читает символы с входного потока. В этот момент программа застынет в ожидании ввода со стороны пользователя. Когда пользователь введет строку и нажмет **<Enter>**, строка попадет в определенный буфер, и из него уже по одному символу будет читаться в нашем цикле с помощью метода **Read()**.

Следующей строкой кода мы конвертируем код прочитанного символа непосредственно в символ. Метод **ToChar()** класса **Convert** превращает индекса символа в символ, который и возвращается в качестве результата. Далее, выводим прочитанное на экран в отдельной строке, и если прочитанный символ не является буквой **q**, то цикл продолжается.

Тут нужно заметить, что выполнение программы прервется не только после того, как пользователь введет отдельно стоящую букву **q**, а после любого слова, в котором есть эта буква, т.е. введя **flaq**, программа выведет четыре символа слова и, увидев **q**, завершит выполнение цикла.

Метод **ReadLine()** тоже читает данные с входного буфера, но он возвращает буфер целиком в виде строки. Следующий пример показывает, как можно читать данные с консоли, пока пользователь не введет букву **q**.

```
String Str;
do {
    Str = Console.ReadLine();
    Console.WriteLine(Str);
   } while (Str != “q”);
```

На этот раз для выхода нужно ввести в строке только q и ничего больше, потому что для выхода из цикла ищется не просто символ q во введенной строке, а сравнивается полученная строка целиком с буквой **q**.

Так как исходные данные вводятся в виде текстовых строк, то при вводе числовых значений необходимо выполнить преобразование строки в число. Для преобразования введенной строки в данные необходимо использовать соответствующие методы преобразования: **Convert.ToInt16(), Convert.ToInt32(), Convert.ToSingle(), Convert.ToDouble()** и т.д.(см. табл. 2), в которые в качестве параметра передается строка ввода.

В процессе преобразования строки в число возможны ошибки (исключения), например, из-за того, что при вводе дробного числа пользователь введет точку вместо запятой ("правильным" символом, при стандартной для Украины настройке операционной системы, является **запятая**).

Таблица 2. Функции преобразования строк

Функция	Значение
ToSingle(s)	Дробное типа Single, Double
ToDouble(s)	
ToByte(s)	Целое типа Byte, Int16, Int32, Int64
ToInt16(s)	
ToInt32(s)	
ToInt64(s)	
ToUInt16(s)	Целое типа Unt16, Unt32, UInt64
ToUInt32(s)	
ToUInt64(s)	

1.2. Базовый вывод с помощью класса Console

Если данные встроенного типа (например, int или double) требуется представить в форме, удобной для восприятия человеком, необходимо создать их строковое представление. Несмотря на то что C# автоматически поддерживает стандартный формат для такого представления, у программиста есть возможность задать собственный формат. Для числовых типов предусмотрен ряд методов форматирования данных, в том числе и методы Console.WriteLine(), String.Format() и ToString().

1.2.1. Общее представление о форматировании

Форматирование реализуется двумя компонентами: спецификаторами и поставщиками (провайдерами) формата. Форма, которую принимает строковое представление форматируемого значения, определяется применяемым спецификатором формата. Другими словами, внешний вид этой формы зависит именно от спецификатора формата. Например, чтобы вывести числовое значение с использованием экспоненциального представления чисел (в виде мантиссы и порядка), необходимо использовать спецификатор формата E.

Во многих случаях на точный формат значения может оказывать влияние территориальное образование или естественный язык, который используется в данной программе. Например, в США денежные суммы представляются в долларах, а в Европе —в евро. Для отображения языковых различий в C# используются поставщики формата.

Поставщик формата определяет способ интерпретации спецификатора формата и создается посредством реализации интерфейса IFormatProvider. Поставщики формата заданы для встроенных числовых типов и многих других типов среды .NET Framework.

Чтобы отформатировать данные с помощью метода WriteLine(), используется метод WriteLine(), имеющий следующий вид:

WriteLine("<строка_форматирования>", arg0, arg1, ... , argN);

где элемент <строка_форматирования> содержит две составляющих: "постоянную" и "переменную". Постоянная составляющая представляет собой печатные символы, отображаемые "как есть", а переменная состоит из команд форматирования, например, оператор

Console.WriteLine("Стоимость обучения {0} гривень", 5400);

где в первом параметре указана строка, в которой есть ссылка на нулевой аргумент — {0}, нулевой аргумент — это второй параметр, выводит на экран строку:

Стоимость обучения 5400 гривень

Общая форма записи команд форматирования имеет такой вид

{<номер_аргумента>, <ширина>: <формат>}

где элемент **<номер_аргумента>** определяет порядковый номер отображаемого аргумента (начиная с нулевого);

элемент **<ширина>** указывает минимальную ширину поля;

спецификатор **<формат>** задается элементом формат.

Например, оператор

Console.WriteLine("Значение z = {0, 10:f2}", 71.5439);

выведет строку «**Значение z = 71,54**»

Если при выполнении метода WriteLine() в строке форматирования встречается команда форматирования, вместо нее подставляется (и отображается) аргумент, соответствующий заданному элементу <номер_аргумента>. Таким образом, элементы <номер_аргумента> указывают позицию спецификации в строке форматирования, которая определяет, где именно должны быть отображены соответствующие данные.

Для форматирования данных необходимо включить в вызов соответствующего метода желаемый спецификатор формата. Если в строке форматирования присутствует элемент <формат>, то соответствующие данные отображаются с использованием заданного формата. В противном случае используется стандартный формат. В этом случае команда форматирование имеет вид

{<номер_аргумента>: <формат>}.

Например, оператор

Console.WriteLine("{0:D12}", 602);

выведет строку «**000000000602**».

При наличии элемента <ширина> выводимые данные дополняются пробелами, которые гарантируют, что поле, занимаемое выводимым значением, будет иметь минимальную ширину. Если значение <ширина> положительно, выводимые данные выравниваются по правому краю, а если отрицательно — по левому.

Спецификаторы формата, определенные для числовых данных, описаны в табл. 3. Каждый спецификатор формата может включать необязательный спецификатор точности Например, чтобы указать, что значение, представляемое в формате с плавающей точкой, должно иметь два десятичных разряда, используйте спецификатор F2.

Элементы ширина и формат указывать необязательно.

Таким образом, в простейшей форме команда форматирования лишь означает, какой аргумент нужно отобразить. Следовательно, команда {0} означает arg0, {1} означает argl и т.д.

Таблица 3. Форматы представления чисел

Параметр	Формат	Пример
c, C	финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие на стройки операционной системы	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, D	числовой. Используется для представления целочисленных цифр с необязательным отрицательным знаком. Поддерживается только целочисленными типами данных.	1234 ("D") => 1234 -1234 ("D6") => -001234
e, E	научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в на стройках операционной системы	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
f, F	число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел опреде ляют соответствующие настройки операционной системы	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
n, N	числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрица тельных чисел определяют соответствующие настройки	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

	операционной системы	
g, G	универсальный формат. Наиболее компактная запись из двух вариантов: экспоненциального и с фиксированной запятой. Поддерживается всеми числовыми типами данных.	-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
r, R	без округления. В отличие от формата N, этот формат не выполняет округления (количество цифр дроб ной части зависит от значения числа)	123456789.12345678 ("R") => 123456789.12345678 -1234567890.12345678 ("R") => -1234567890.1234567
x, X	Числовой. Используется для представления чисел в шестнадцатеричном виде. Поддерживается только целочисленными типами данных.	255 ("X") => FF -1 ("x") => ff 255 ("x4") => 00ff -1 ("X4") => 00FF

После выполнения инструкции **Write()** курсор остается в той позиции экрана, в которую он переместился после вывода последнего символа строки. Сле дующая инструкция **Write()** начинает вывод с той позиции экрана, в которой находится курсор.

Метод **WriteLine()** отличается от метода **Write()** тем, что после вывода строки курсор автоматически переходит в начало следующей строки. Суще ствует множество вариантов этого метода, боль шинство которых — это вариации на тему типа данных, получаемых в параметре. Метод может принимать числа, булевы значения и т. д., переводить их в строку и выводить в окно консоли.

Пример 2. В качестве примера создадим консольное приложение, которое демонстрирует использование ряда спецификаторов формата для представления числовых данных:

```
using System;
class FormatDemo {
public static void Main()
    {
        double v = 17688.65849;
        int x = 21;
        Console.WriteLine("{0:F2}", v);
        Console.WriteLine("{0:N5}", v);
        Console.WriteLine("{0:e}", v);
        Console.WriteLine("{0:r}", v);
        Console.WriteLine("{0:X}", x);
        Console.WriteLine("{0:D12}", x);
        Console.WriteLine("{0:C}", 189.99);
    }
```

Результатом выполнения этой программы является:

```
17688,66
17 688.65849
1,768866e+004
17688,65849
15
000000000021
189,99р.
```

В строке форматирования можно использовать специальные символы фор матирования (например, **\n** — переход на новую строку; **\r** — переход в начало строки). Оператор

```
Console.WriteLine("Первое число {0} \n Второе число {1}", 25, 35);
```

выводит с помощью одного метода сразу две строки. Специальный символ **\n** будет заменен на переход на новую строку. В результате на экране монитора будет:

Первое число 25
Второе число 35

1.2.2. Использование метода ToString() для форматирования данных

Несмотря на то, что встраивание команд форматирования в метод WriteLine() —очень удобный способ вывода данных в сформатированном виде, иногда имеет смысл ограничиться созданием строки, которая будет содержать форматированные данные, т.е. в некоторых ситуациях нужно сформатировать данные заранее, чтобы они были готовы к последующему выводу на любое заданное устройство.

Для получения форматированного нужным образом строкового представления, соответствующего значению встроенного числового типа (например, Int32 или Double), можно использовать метод ToString():

public string ToString(string fmt)

Метод ToString() возвращает строковое представление вызывающего объекта в соответствии с заданным спецификатором формата, переданным в параметре fmt (**"c"** — финансовый; **"n"** — числовой (см. табл. 3)). Например, оператор

Console.Write ("Значение переменной a равно " + a.ToString("f"));

выведет на экран монитора значение переменной **a** в виде числа с фиксированной точкой.

В отличие от метода WriteLine(), который использует встроенные команды форматирования (вместе с номером аргумента и значением ширины поля), метод ToString() принимает только спецификатор формата.

Пример 3. Вот как выглядит новая версия предыдущей программы форматирования, которая для форматирования строк использует метод ToString(). Результаты выполнения новой версии совпадают с результатами предыдущей.

```
using System;
class ToStringDemo {
public static void Main()
    {
      double v = 17688.65849;
      int x = 21;
      string str = v.ToString("F2");
      Console.WriteLine(str);
      str = v.ToString("N5");
      Console.WriteLine(str);
      str = v.ToString("e");
      Console.WriteLine(str);
      str = v.ToString("r");
      Console.WriteLine(str);
      str = x.ToString("X");
      Console.WriteLine(str);
      str = x.ToString("D12");
      Console.WriteLine(str);
      str = 189.99.ToString("C") ;
      Console.WriteLine(str);
    }
```

1.2.3. Создание пользовательского числового формата

Несмотря на то что встроенные спецификаторы формата весьма полезны, в C# программист может определить собственный формат, используя средство, называемое форматом изображения (picture format). Своим названием этот термин обязан тому факту, что программист создает формат на

основе примера (т.е. изображения) того, как должен выглядеть выводимый результат.

При создании пользовательского формата задается пример, или изображение того, как должны выглядеть данные. Для этого используются символы, приведенные в табл. 4.

Таблица 4. Символы-заполнители специального формата числовых данных

Заполнитель	Описание
#	Цифра. Символ "# " задает позицию цифры и может располагаться слева или справа от десятичной точки или без таковой. Если этот символ (один или несколько) находится справа от десятичной точки, он (они) означает количество отображаемых десятичных разрядов. При необходимости значение округляется. Если символ "# " находится слева от десятичной точки, он указывает позиции цифр, относящиеся к целой части числа. При необходимости добавляются начальные нули. Если целая часть числа содержит больше цифр, чем в формате представлено символов "#", такая целая часть числа отображается полностью и ни в коем случае не усекается. Если десятичная точка отсутствует, наличие символа "# " означает, что форматируемое значение будет округлено до целого числа. Нулевое значение (без указания дополнить его начальными или концевыми нулями) отображаться не будет. Это значит, что при использовании такого формата, как ##.##, ничего не будет отображено, если форматируемое значение равно нулю. Чтобы отобразить таки нулевое значение, используйте символ-заполнитель "0".
.	Десятичная точка. Символ "точка" указывает местоположение десятичной точки.
,	Разделитель групп разрядов. При форматировании больших чисел можно заказать вставку разделителей групп разрядов, задав шаблон отображаемого значения в виде последовательности символов "#" со вставленной в нее запятой. Например, при выполнении инструкции **Console.WriteLine (" { 0 : #,###.# } " , 3421.3) ;** отображается такой результат: **3,421.3.** Обратите здесь внимание на то, что нет необходимости вставлять символ "запятая" во все предполагаемые позиции. Задания в шаблоне лишь одной запятой уже вполне достаточно для вставки ее в значение после каждых трех цифр (начиная от десятичной точки) в левой части числа. Например, при выполнении инструкции **Console.WriteLine ("{ 0 : #,###.# }" , 8763421.3) ;** отображается следующий результат: **8,763,421.3.** Символы "запятая" в строке форматирования имеют и еще одно значение. Если они стоят слева от десятичной точки (но рядом с ней), то действуют как масштабный множитель. Каждая запятая обеспечивает деление форматируемого значения на 1 000. Например, при выполнении инструкции **Console.WriteLine("Значение в тысячах: {0:#,###,.#}", 8763421.3);** отображается такой результат: **Значение в тысячах: 8,7 63.4**
%	Процент
0	Используется для дополнения начальными и концевыми нулями. Благодаря символу-заполнителю "0" выводимое значение дополняется начальными или концевыми нулями, гарантирующими наличие минимального количества цифр, заданного строкой форматирования. Этот символ можно использовать как с правой, так и с левой стороны от десятичной точки. Например, при выполнении инструкции **Console.WriteLine("{0:00##.#00}", 21.3);** отображается такой результат: **0021.300**
;	Символ ";" представляет собой разделитель, который позволяет задать различные форматы для отображения положительных, отрицательных и нулевых значений. Общая форма записи спецификатора формата с использованием символа-заполнителя ";" такова:

	плюс-формат; минус-формат; нуль-формат Например, **Console. WriteLine ("{0 :#.##; (#.##) ;0.00}", num) ;** Если значение num положительно, при выводе оно будет отображаться с двумя десятичными разрядами. Если num отрицательно, то при выводе оно будет отображено с двумя десятичными разрядами и заключено в круглые скобки. Если num равно нулю, отобразится строка 0.00. При использовании символа-заполнителя " ; " необязательно задавать все три части формата. Если важно указать, как должны выглядеть при выводе положительные и отрицательные значения, опустите элемент нуль-форма т. Чтобы для отображения отрицательных значений использовать стандартный формат, опустите элемент минус-форма т. В этом случае элементы плюс-формат и нуль-формат необходимо разделить двумя символами "точка с запятой".
E0 E+0 E-0 **e0 e+0 e-0**	Экспоненциальное представление чисел. Символы-заполнители E и e означают, что форматируемое значение должно быть отображено в экспоненциальном представлении (в виде мантиссы и порядка). После символа E и e должен стоять хотя бы один символ "0". Количество символов "0" означает число десятичных цифр, которые должны отображаться при выводе. Дробная часть округляется в соответствии с заданным форматом. Прописному написанию символа-заполнителя соответствует отображение прописной буквы E, а строчному — строчная (e). Для гарантированного отображения знака порядка используйте формат E+ или e+, а для отображения знака только в случае отрицательного порядка используйте один из таких форматов: E, e, E- или e-.

Помимо символов-заполнителей пользовательский спецификатор формата может содержать и другие символы. Любые символы, отличные от заполнителей, без изменений отображаются в отформатированной строке, причем в соответствующих шаблону позициях. Например, при выполнении инструкции

Console.WriteLine("Топливная экономичность равна {0:##.# км на литр}", 21.3);

отображается такой результат:

Топливная экономичность равна 21.3 км на литр

Пример 4. В следующей программе демонстрируется использование нескольких из многочисленных пользовательских форматов, которые может создать программист:

```
using System;
class PictureFormatDemo {
public static void Main{)
    {
     double num = 64354.2345;
     Console.WriteLine("Стандартный формат: " + num);
     // Отображаем значение с двумя десятичными разрядами
     Console.WriteLine("Значение с двумя десятичными разрядами: " +
                       "{0:#.##}", num);
     // Отображаем значение с запятыми и двумя десятичными разрядами
     Console.WriteLine("Добавляем запятые: {0:#,###.##}", num) ;
     // Отображаем значение в экспоненциальном представлении
     Console.WriteLine("Используем экспоненциальное представление: " +
                       "{0:#.###e+00}", num);
     // Отображаем значение в тысячах
     Console.WriteLine("Значение в тысячах: " + "{0:#0,}", num);
```

```
        // Отображаем положительные, отрицательные и нулевые значения по-разному
        Console.WriteLine("Отображаем положительные, " +
                          "отрицательные и нулевые  " + "значения по-разному.");
         Console.WriteLine("{0:#.#; (#.##); 0.00}", num) ;
         num = -num;
         Console.WriteLine("{0:#.##;(#.##);0.00}", num);
         num =0.0;
         Console.WriteLine("{0:#.##; (#.##); 0 . 00}", num);
       }
```

Результаты выполнения этой программы таковы:

Стандартный формат: 64354.2345
Значение с двумя десятичными разрядами: 64354.23
Добавляем запятые: 64,354.23
Используем экспоненциальное представление: 6.435e+04
Значение в тысячах: 64
Отображаем положительные, отрицательные и нулевые значения по-разному.
64354.2
(64354.23)
0.00

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по вводу-выводу данных и выполнить практически все примеры, описанные в этом разделе.

Задание 2. Разработать приложение с заголовком «Задание №2», которое выводит на экран монитора фамилию, имя, отчество и номер учебной группы студента. Ввод данных осуществляется с клавиатуры.

Задание 3. Разработать приложение с заголовком «Задание №3», с помощью которого можно вводить символ и два числа (целого типа и вещественного типа с двойной точностью) и выводить соответствующие сообщения о значениях переменных (один из вариантов решения представлен на рис. 1).

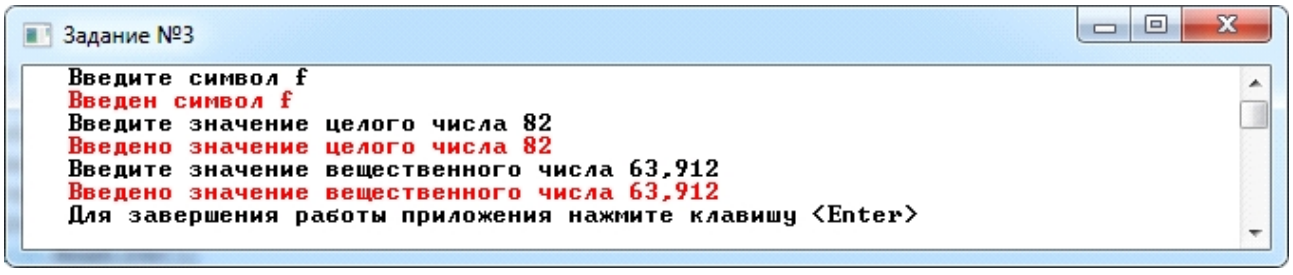

Рис. 1. Вариант консольного приложения «Задание №3»

Дополнительное задание. Разработать приложение с заголовком «Бегущая строка», с помощью которого можно вводить однострочный текст, который выводить на экран монитора в виде бегущей строки. Для определения темпа изменения местоположения строки использовать оператор

System.Threading.Thread.Sleep(1000);

Для определения местоположения начала строки использовать оператор

Console.SetCursorPosition(left, top);

где **left** – имя переменной, в которой хранится значение позиции столбца курсора (столбцы нумеруются слева направо начинается с 0), **top** – имя переменной, в которой хранится значение позиции строки курсора (строки нумеруются сверху вниз начинается с 0).

3. Контрольные вопросы

1. Перечислите методы и дайте их краткую характеристику, с помощью которых осуществляется ввод данных в консольном приложении.
2. Перечислите методы и дайте их краткую характеристику, с помощью которых осуществляется вывод данных в консольном приложении.
3. Перечислите методы и дайте их краткую характеристику, с помощью которых осуществляется преобразование строк в число.
4. Перечислите форматы представления чисел и дайте их краткую характеристику.
5. Дайте характеристику пользовательскому формату.

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011.– 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.