Лабораторная работа №1

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ МЕХАТРОНИКИ ТРАНСПОРТНЫХ СРЕДСТВ

Кафедра информатики

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению практических работ по дисциплине «Программирование» для студентов специальности 6.050201 “Системная инженерия”

Разработчик - доцент кафедры информатики
кандидат технических наук,
старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2012

Лабораторная работа №1
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений, обрабатывающие строки.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, обрабатывающие строки.

1. Теоретические сведения

Во всех языках программирования строка (string) представляет собой последова тельность символов, но точная реализация такой последовательности меняется при переходе от одного языка к другому. Строки в C# — это объекты встроенного типа данных string, поэтому string является ссылочным типом данных. Более то го, string — это C#-имя стандартного строкового типа .NET-среды System.String. C#-строка имеет доступ ко всем методам, свойствам, полям и опера торам, определенным в классе String.

1.1. Конструкторы класса String

Класс String определен в пространстве имен System. В классе String определено несколько конструкторов, которые позволяют созда вать строки различными способами:

public String(char [] *str*)

public String(char [] *str*, int *start*, int *count*)

Первый формат предназначен для построения строки, которая будет состоять из символов, содержащихся в массиве ***str***. Строка, создаваемая с помощью второго формата, будет состоять из ***count*** символов, взятых из массива ***str***, начиная с сим вола, индекс которого задан параметром ***start***.

Существует возможность создать строку, содержащую заданный символ, по вторенный нужное количество раз. Для этого используйте этот конструктор:

public String(char *ch*, int *count*)

где параметр ***ch*** задает символ, который будет повторен ***count*** раз.

Используя один из следующих конструкторов, можно создать строку, заданную указателем на символьный массив:

unsafe public String(char* *str*)

unsafe public String(char* *str*, int *start*, int *count*)

Конструктор первого формата предназначен для построения строки, содержащей символы, на которые указывает параметр ***str*** (пара метр ***str*** указывает на массив с завершающим нулем (символом конца строки)). Строка, создаваемая с помощью конструктора второго формата, будет состоять из ***count*** символов, взятых из массива, адресуемого указателем ***str***, начиная с символа, индекс которого задан параметром ***start***.

Используя один из следующих конструкторов, можно создать строку, заданную указателем на массив байтов:

unsafe public String(sbyte* *str*)

unsafe public String(sbyte* *str*, int *start*, int *count*)

unsafe public String(sbyte* *str*, int *start*, int *count*, Encoding *en*)

Конструктор первого формата предназначен для построения строки, содержащей байты, на которые указывает параметр ***str*** (параметр ***str*** указывает на массив с завершающим нулем (символом конца строки)). Строка, создаваемая с помощью конструктора второго формата, будет состоять из ***count*** бай тов, взятых из массива, адресуемого указателем ***str***, начиная с байта, индекс кото рого задан параметром ***start***. Третий формат конструктора позволяет указать тип ко дирования байтов. По умолчанию используется тип ASCIIEncoding. Класс Encoding определен в пространстве имен System.Text.

Строковый литерал создает строковый объект автоматически. Поэтому строковый объект часто инициализируется присваиванием ему строкового литерала, например:

string str = "новая строка";

Основным свойством объекта **String** является **Length**, которое возвращает число символов в строке. Например, оператор **n = Str.Length;** записывает в переменную **n** количество символов в строке **Str**.

1.2. Методы класса String

Основными методами класса **String** являются:

- Compare() – сравнивает две строки **String** целиком или по частям, причем с учетом (или без) прописного или строчного варианта букв (т.е. регистра клавиатуры). В общем случае при сравнении строк, т.е. при определении того, больше ли одна строка другой, меньше или они равны, используется лексикографический по рядок. В классе String предусмотрен широкий выбор методов сравнения, которые перечислены в табл.1.

Синтаксис метода **public static int Compare(string *strA*, string *strB*),**
где ***strA*** - первая сравниваемая строка, ***strB*** – вторая сравниваемая строка. Возвращаемое значение - знаковое целое число, выражающее лексическое отношение двух сравниваемых значений (возвращает положительное чис ло, если строка ***strA*** больше ***strB***, отрицательное число, если ***strA*** меньше ***strB***, и нуль, если строки ***strA*** и ***strB*** равны).

Таблица 1. Методы сравнения, определенные в классе String

Метод	Описание
public static int Compare(string *strA*, string *strB*)	Сравнивает строку, адресуемую параметром ***strA***, со строкой, адресуемой параметром ***strB***
public static int Compare(string *strA*, string *strB*, bool *ignoreCase*)	Сравнивает строку, адресуемую параметром ***strA***, со строкой, адресуемой параметром ***strB***. Если параметр ***ignoreCase*** равен значению **true**, при срав нении не учитываются различия между прописным и строчным вариантами букв. В противном случае эти различия учитываются
public static int Compare(string *strA*, string *strB*, bool *ignoreCase*, CultureInfo *ci*)	Сравнивает строку, адресуемую параметром ***strA***, со строкой, адресуемой параметром ***strB***, с использованием специальной информации (связанной с конкретным естественным языком, диалектом или территориальным образованием), переданной в параметре ***ci***. Если пара метр ***ignoreCase*** равен значению **true**, при сравнении не учитываются различия между прописным и строчным варианта ми букв. В противном случае эти различия учитываются. Класс CultureInfo определен в пространстве имен System.Globalization.
public static int Compare(string *strA*, int *start1*, string *strB*, int *start2*, int *count*)	Сравнивает части строк, заданных параметрами ***strA*** и ***strB***. Сравнение начинается со строковых элементов ***strA[start1]*** и ***strB[start1]*** и включает ***count*** символов. Метод воз вращает положительное число, если часть строки ***strA*** больше части строки ***strB***, отрицательное число, если часть ***strA*** меньше части ***strB***, и нуль, если сравниваемые части строк ***strA*** и ***strB*** равны.
public static int Compare(string *strA*, int *start1*, string *strB*, int *start2*, int *count*, bool *ignoreCase*)	Сравнивает части строк, заданных параметрами ***strA*** и ***strB***. Сравнение начинается со строковых элементов ***strA[start1]*** и ***strB[start1]*** и включает ***count*** символов. Метод воз вращает положительное число, если часть строки ***strA*** больше части строки ***strB***, отрицательное число, если часть ***strA*** меньше части ***strB***, и нуль, если сравниваемые части строк ***strA*** и ***strB*** равны. Если параметр ***ignoreCase*** равен значению **true**, при срав нении не учитываются различия между прописным и строчным вариантами букв. В

	противном случае эти различия учитываются.
public static int Compare(string *strA*, int *start1*, string *strB*, int *start2*, int *count*, bool *ignoreCase*, CultureInfo *ci*)	Сравнивает части строк, заданных параметрами ***strA*** и ***strB***, с использованием специальной информации (связанной с конкретным естественным языком, диалектом или территориальным образованием), переданной в параметре ***ci***. Сравнение начинается со строковых элементов ***strA[start1]*** и ***strB[start1]*** и включает ***count*** символов. Метод воз вращает положительное число, если часть строки ***strA*** больше части строки ***strB***, отрицательное число, если часть ***strA*** меньше части ***strB***, и нуль, если сравниваемые части строк ***strA*** и ***strB*** равны. Если параметр ***ignoreCase*** равен значению **true**, при срав нении не учитываются различия между прописным и строчным вариантами букв. В противном случае эти различия учитываются. Класс CultureInfo определен в пространстве имен System.Globalization.
public static int CompareOrdinal(string *strA*, string *strB*)	Сравнивает строку, адресуемую параметром ***strA***, со строкой, адресуемой параметром ***strB***, независимо от языка, диалекта или территориального образования.
public static int CompareOrdinal(string *strA*, int *start1*, string *strB*, int *start2*, int *count*)	Сравнивает части строк, заданных параметрами ***strA*** и ***strB***, независимо от языка, диалекта или территориального образо вания. Сравнение начинается со строковых элементов ***strA[start1]*** и ***strB[start1]*** и включает ***count*** символов. Метод возвращает положительное число, если часть строки ***strA*** больше части строки ***strA***, отрицательное чис ло, если часть ***strA*** меньше части ***strB***, и нуль, если срав ниваемые части строк равны.
public int CompareTo(object *str*)	Сравнивает вызывающую строку со строкой, заданной пара метром ***str***. Возвращает положительное число, если вызы вающая строка больше строки ***str***, отрицательное число, если вызывающая строка меньше строки ***str***, и нуль, если сравни ваемые строки равны.
public int CompareTo(string *str*)	Сравнивает вызывающую строку со строкой, заданной пара метром ***str***. Возвращает положительное число, если вызы вающая строка больше строки ***str***, отрицательное число, если вызывающая строка меньше строки ***str***, и нуль, если сравни ваемые строки равны.

Пример 1. Нижеприведенный код сравнивает две стоки:

```
private void button1_Click(object sender, EventArgs e)
        {
          if (String.Compare(textBox1.Text, textBox2.Text) == 0)
            label1.Text = textBox1.Text + " и " + textBox2.Text + " равны";
           else if (String.Compare(textBox1.Text, textBox2.Text) > 0)
            label1.Text = textBox1.Text + " больше " + textBox2.Text;
              else label1.Text = textBox1.Text + " меньше " + textBox2.Text;
        }
```

- Concat(string, string) – сцепляет две строки.

Синтаксис метода **public static string Concat(string *strA*, string *strB*),** где ***strA*** - первая сравниваемая строка, ***strB*** – вторая сравниваемая строка. Возвращаемое значение – сцепление строки ***strA*** и ***strB***.

Пример 2. Нижеприведенный код выводит сцепку строк **FIO** в поле вывода:

```
private void button1_Click(object sender, EventArgs e)
        {
```

```
            string Fam = "Иванов";
            string Name = "Сергей";
            string Otch = "Петрович";
            string FIO;
            Name = String.Concat(  " ", Name.Trim() );
            Otch = String.Concat(  " ", Otch.Trim() );
            FIO = String.Concat( String.Concat( Fam, Name ), Otch );
            textBox1.Text = FIO;
        }
```

- IndexOf(char) - возвращает индекс первого вхождения указанного знака Юникода в данной строке.

Синтаксис метода **public int IndexOf(char *ch*),**
где ***ch*** - искомый знак Юникода. Возвращаемое значение – значение индекса (начиная с нуля) параметра ***ch***, если этот символ найден, или значение **-1**, если он не найден.

- IndexOf(string) - возвращает индекс первого вхождения значения указанной строки в данном экземпляре.

Синтаксис метода **public int IndexOf(string *str*)**
где ***str*** - строка для поиска. Возвращаемое значение – значение индекса (начиная с нуля) параметра ***str***, если эта строка найдена, или значение **-1**, если она не найдена.


Пример 3. Нижеприведенный код демонстрирует поиск первого вхождения строки

```
private void button1_Click(object sender, EventArgs e)
        {
            String Str = "Это моё приложение";
            String Str1 = "моё";
            textBox1.Text = Str.IndexOf(Str1).ToString("d");
        }
```

- IndexOf(char, Int32) - возвращает индекс первого вхождения указанного знака Юникода в данной строке.

Синтаксис метода **public int IndexOf(char *ch*, int *startIndex*),**
где ***ch*** - искомый знак Юникода, ***startIndex*** - позиция, с которой начинается поиск. Параметр ***startIndex*** может иметь значение в диапазоне от **0** до величины, равной длине экземпляра строки минус **1**. Диапазон поиска от **startIndex** до конца строки. Возвращаемое значение - значение индекса (начиная с нуля) параметра ***ch***, если этот символ найден, или значение **-1**, если он не найден.

- IndexOf(string, Int32) - возвращает индекс первого вхождения значения указанной строки в данном экземпляре.

Синтаксис метода **public int IndexOf(string *Str*, int *startIndex*),**
где ***Str*** - строка для поиска, ***startIndex*** - позиция, с которой начинается поиск. Параметр ***startIndex*** может иметь значение в диапазоне от **0** до величины, равной длине экземпляра строки минус **1**.Возвращаемое значение - положение в индексе (начиная с нуля) параметра ***Str***, если эта строка найдена, или значение **-1**, если она не найдена.

- Insert(Int32, string) - вставляет указанный экземпляр **String** в данный экземпляр по заданному индексу.

Синтаксис метода **public String Insert(int *startIndex*, string *Str*),**
где ***startIndex*** - индекс позиции вставки, ***Str*** - строка, которую требуется вставить. Возвращаемое значение - новая строка, эквивалентная данному экземпляру, но с тем отличием, что в позицию **startIndex** помещено значение ***Str***. Если ***startIndex*** равен длине данного экземпляра, то ***Str*** добавляется к концу этого экземпляра.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой ***Str*** вставляется в текущий экземпляр.

Пример 4. Нижеприведенный код демонстрирует вставку строки в другую строку

```
private void button1_Click(object sender, EventArgs e)
        {
          string Str1 = "Студенты изучают математику, физику";
          string Str2 = "программирование";
          Str2 = String.Concat( Str2.Trim(), " " );
          textBox1.Text = Str1.Insert(17, Str2);
        }
```

- LastIndexOf(char) - возвращает индекс последнего вхождения указанного знака Юникода в пределах данного экземпляра.

Синтаксис метода **public int LastIndexOf(char *ch*),**
где ***ch*** - знак Юникода, который необходимо найти. Возвращаемое значение - положение в индексе (начиная с нуля) параметра ***ch***, если этот символ найден, или значение **-1**, если он не найден.

Этот метод начинает поиск с позиции последнего знака данного экземпляра и выполняет его от конца к началу, пока не обнаружит ***ch*** либо не дойдет до позиции первого знака. При поиске учитывается регистр.

- LastIndexOf(string) - возвращает значение индекса последнего вхождения указанной строки в данном экземпляре.

Синтаксис метода **public int LastIndexOf(string *Str*),**
где ***Str*** - строка для поиска. Возвращаемое значение - положение в индексе (начиная с нуля) параметра ***Str***, если эта строка найдена, или значение **-1**, если она не найдена.

Поиск начинается с позиции последнего знака данного экземпляра и выполняется в обратном направлении к началу, пока не обнаружится ***Str*** либо не будет проверен первый знак.

- LastIndexOf(char, Int32) - возвращает индекс последнего вхождения указанного знака Юникода в пределах данного экземпляра. Поиск начинается с указанной позиции знака.

Синтаксис метода **public int LastIndexOf(char *ch*, int *startIndex*),**
где ***ch*** - знак Юникода, который необходимо найти, ***startIndex*** - начальная позиция подстроки в пределах данного экземпляра. Возвращаемое значение - положение в индексе (начиная с нуля) параметра ***ch***, если этот символ найден, или значение **-1**, если он не найден.

Этот метод начинает поиск с позиции знака ***startIndex*** данного экземпляра и выполняет его от конца к началу, пока не обнаружит ***ch*** либо не дойдет до позиции первого знака. Например, если параметр ***startIndex*** равен значению свойства **Length** минус **1**, метод выполняет поиск каждого знака, от последнего знака строки до начального. При поиске учитывается регистр.

- LastIndexOf(string, Int32) - возвращает значение индекса последнего вхождения указанной строки в данном экземпляре. Поиск начинается с указанной позиции знака.

Синтаксис метода **public int LastIndexOf(string *Str*, int *startIndex*),**
где ***Str*** - строка для поиска, ***startIndex*** - позиция, с которой начинается поиск.

Возвращаемое значение - положение в индексе (начиная с нуля) параметра ***Str***, если эта строка найдена, или значение **-1**, если она не найдена.

Поиск начинается с позиции знака ***startIndex*** данного экземпляра и выполняется в обратном направлении к началу, пока не обнаружится ***Str*** либо не будет проверен первый знак. Например, если параметр ***startIndex*** равен значению свойства **Length** минус **1**, метод выполняет поиск каждого знака, от последнего знака строки до начального. Этот метод выполняет поиск по словам (с учетом регистра и языка и региональных параметров), используя текущий язык и региональные параметры.

Пример 5. Нижеприведенный код демонстрирует поиск индекса всех вхождений строки в целевой строке от конца целевой строки до ее начала.

```
private void button1_Click(object sender, EventArgs e)
        {
          string str = "Не ной и не носись со своими неудачами, учись на них.";
```

```
int start;
start = str.Length - 1;
int pos = 0;
while ((start > -1) && (pos > -1))
{
   pos = str.LastIndexOf("но", start);
   if (pos > -1)
   {
    textBox1.Text = textBox1.Text +"Индекс = "+ pos.ToString("d") + "  ";
    start = pos - 1;
   }
}
```

- Remove(Int32) - удаляет из данной строки все знаки начиная с заданной и до последней позиции.

Синтаксис метода **public string Remove(int *startIndex*),**
где ***startIndex*** - позиция (начиная с нуля), с которой начинается удаление знаков. Возвращаемое значение - новая строка, эквивалентная данной строке за минусом удаленных знаков.

Данный метод не изменяет значение текущего экземпляра. Вместо этого он возвращает новую строку, в которой все символы от позиции ***startIndex*** до конца исходной строки были удалены.

- Remove(Int32, Int32) - удаляет заданное число знаков из данного экземпляра начиная с указанной позиции.

Синтаксис метода **public String Remove(int *startIndex*, int *count*),**
где ***startIndex*** - позиция (начиная с нуля), с которой начинается удаление знаков, ***count*** - число символов для удаления. Возвращаемое значение - новая строка, эквивалентная данному экземпляру за минусом удаленных знаков.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой удалено количество символов, указанное в параметре ***count***. Символы удаляются в позиции, указанной в параметре ***startIndex***.

Пример 6. Нижеприведенный код демонстрирует удаление имени из полного имени

```
private void button1_Click(object sender, EventArgs e)
      {
       string name = "Кравченко Виктор Семенович";
       int foundS1 = name.IndexOf( " " );
       int foundS2 = name.IndexOf( " ", foundS1 + 1 );
       if ( foundS1 != foundS2 && foundS1 >= 0 )
        {
         name = name.Remove( foundS1 + 1, foundS2 - foundS1 );
         textBox2.Text = name;
        }
      }
```

- Replace(char, char) - возвращает новую строку, в которой все вхождения заданного знака Юникода в текущем экземпляре заменены другим заданным знаком Юникода

Синтаксис метода **public string Replace(char *old_ch*, char *new_ch*),**
где ***old_ch*** - заменяемый знак Юникода, ***new_ch*** - знак Юникода для замены всех обнаруженных вхождений ***old_ch***. Возвращаемое значение - строка, эквивалентная данному экземпляру, но с тем отличием, что все вхождения ***old_ch*** заменены на ***new_ch***.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой все вхождения ***old_ch*** заменяются на ***new_ch*.**

- Replace(string, string) - возвращает новую строку, в которой все вхождения заданной строки в текущем экземпляре заменены другой заданной строкой.

Синтаксис метода **public string Replace(string *old_Str*, string *new_Str*),**
где ***old_Str*** - строка, которую требуется заменить, ***new_Str*** - строка для замены всех вхождений ***old_Str***. Возвращаемое значение - строка, эквивалентная текущей строке, но с тем отличием, что все вхождения ***old_Str*** заменены на ***new_Str***. Если ***new_Str*** равно **nullptr**, все вхождения ***old_Str*** удаляются.


Пример 7. Нижеприведенный код демонстрирует замену строк (используется для исправления орфографических ошибок).

```
private void button1_Click(object sender, EventArgs e)
      {
       string errStr = "При работе приложения возникают ашибки, связанные
                                                  с данными, ашибками пользователя";
       string correctStr = errStr.Replace( "ашибк", "ошибк" );
       textBox1.Text = correctStr;
      }
```

- Substring(Int32) - извлекает подстроку из данного экземпляра. Подстрока начинается с указанной позиции знака.

Синтаксис метода **public string Substring(int *startIndex*),**
где ***startIndex*** - позиция первого знака подстроки в данном экземпляре (с нуля). Возвращаемое значение - строка, эквивалентная подстроке, которая начинается с ***startIndex*** в данном экземпляре, или **Empty**, если значение ***startIndex*** равно длине данного экземпляра.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, которая начинается с позиции ***startIndex*** в текущей строке.

- Substring(Int32, Int32) - извлекает подстроку из данного экземпляра. Подстрока начинается с указанной позиции знака и имеет указанную длину.

Синтаксис метода **public string Substring(int *startIndex*, int *length*),**
где ***startIndex*** - позиция первого знака подстроки в данном экземпляре (с нуля), ***length*** - число символов в подстроке. Возвращаемое значение - строка, эквивалентная подстроке длиной ***length***, которая начинается с ***startIndex*** в данном экземпляре, или **Empty**, если значение ***startIndex*** равно длине данного экземпляра, а значение ***length*** равно нулю.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка с ***length*** символами, начиная с позиции ***startIndex*** в текущей строке.


Пример 8. Нижеприведенный код демонстрирует извлечение подстроки из строки.

```
private void button1_Click(object sender, EventArgs e)
      {
         string Str1 = "Кузменко Андрей Иванович";
         string Str2 = Str1.Substring(9, 6);
         textBox1.Text = Str2;
      }
```

- ToLower() - возвращает копию этой строки, переведенную в нижний регистр.

Синтаксис метода **public string ToLower()**. Возвращаемое значение - строка в нижнем регистре.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой все символы текущего экземпляра преобразуются в строчные. Оператор

```
str1 = str.ToLower();
```

выполняется преобразование строки **str** со смешанным регистром знаков в нижний регистр.

- ToUpper() - возвращает копию этой строки, переведенную в верхний регистр.

Синтаксис метода **public string ToUpper()**. Возвращаемое значение - эквивалент текущей строки

в верхнем регистре.

Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой все символы текущего экземпляра преобразуются в прописные.

- **Trim() -** для удаления из строки начальных и концевых пробелов. Используется один из следующих вариантов метода:

public string Trim()

public string Trim(params char[] *chrs*)

Первый формат метода предназначен для удаления начальных и концевых пробелов из вызывающей строки. Второй позволяет удалить начальные и концевые симво лы, заданные параметром ***chrs***. В обоих случаях возвращается строка, содержащая ре зультат этой операции.

- TrimEnd(params char[] chrs) -

удаляет все конечные вхождения набора знаков, заданного в виде массива, из текущего объекта **String**.

Синтаксис метода **public string TrimEnd(params char[]*chrs*),**
где ***chrs*** - массив удаляемых знаков Юникода или **nullptr**. Возвращаемое значение - строка, оставшаяся после удаления всех вхождений знаков, заданных в параметре ***chrs***, из конца текущей строки. Если значением параметра ***chrs*** является **nullptr** или пустой массив, удаляются знаки пробела в Юникоде.

Метод **TrimEnd()** удаляет из текущей строки все конечные знаки, заданные в параметре ***chrs***. Операцию усечения прекращается, когда в конце строки встречается первый знак, не из ***chrs***. Например, если текущей строкой является "123abc456xyz789" и ***chrs*** содержит цифры от "1" до "9", метод **TrimEnd()** возвращает значение "123abc456xyz". Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой все конечные символы, найденные в ***chrs***, удаляются из текущей строки.

- TrimStart(params char[] chrs) -

удаляет все начальные вхождения набора знаков, заданного в виде массива, из текущего объекта **String**.

Синтаксис метода **public string TrimStart(params char[] *chrs*),**
где ***chrs*** - массив удаляемых знаков Юникода или **nullptr**. Возвращаемое значение - строка, оставшаяся после удаления всех вхождений знаков, заданных в параметре ***chrs***, из начала текущей строки. Если значением параметра ***chrs*** является **nullptr** или пустой массив, удаляются знаки пробела.

Метод **TrimStart()** удаляет из текущей строки все начальные знаки, заданные в параметре ***chrs***. Выполнение операции удаления конечных знаков прекращается, когда встречен знак, не содержащийся в ***chrs***. Например, если текущей строкой является "123abc456xyz789" и ***chrs*** содержит цифры от "1" до "9", метод **TrimStart()** возвращает значение "abc456xyz789". Данный метод не изменяет значение текущего экземпляра. Вместо этого возвращается новая строка, в которой удалены все начальные символы пробелов, найденные в текущем экземпляре.

В C# предусмотрена возможность дополнять строку заданными символами справа либо слева. Для реализации "левостороннего" дополнения строки используется один из следующих методов:

public string PadLeft(int *len*)

public string PadLeft(int *len*, char *ch*)

Первый формат метода предназначен для дополнения строки с левой стороны пробелами в таком количестве, чтобы общая длина вызывающей строки стала равной заданному значению ***len***. Второй формат отличается от первого тем, что для допол нения строки вместо пробела используется символ, заданный параметром ***ch***. В обоих случаях возвращается строка, содержащая результат этой операции.

Для реализации "правостороннего" дополнения строки используется один из сле дующих методов:

public string PadRight(int *len*)

public string PadRight(int *len*, char *ch*)

Первый формат метода дополняет строку с правой стороны пробелами в таком ко личестве, чтобы общая длина вызывающей строки стала равной заданному значению ***len***. Второй формат отличается от первого тем, что для дополнения строки вместо пробела используется символ, заданный параметром ***ch***. В обоих случаях возвращает ся строка, содержащая результат этой операции.

2. Рабочее задание

Задание 1. Самостоятельно разработать приложение, которое позволяет продемонстрировать работу методов класса **String**:

- **Compare(string, string);**
- **Concat(string, string);**
- **IndexOf(char);**
- **IndexOf(string);**
- **IndexOf(char, Int32);**
- **IndexOf(string, Int32);**
- **Insert(Int32, string);**
- **LastIndexOf(char);**
- **LastIndexOf(string);**
- **LastIndexOf(char, Int32);**
- **LastIndexOf(string, Int32);**
- **Remove(Int32);**
- **Remove(Int32, Int32);**
- **Replace(char, char);**
- **Replace(string, string);**
- **Substring(Int32);**
- **Substring(Int32, Int32);**
- **ToLower();**
- **ToUpper();**
- **Trim();**
- **TrimEnd(params char[] chrs);**
- **TrimStart(params char[] chrs);**
- **PadLeft(Int32);**
- **PadLeft(Int32, char);**
- **PadRight(Int32);**
- **PadLeft(Int32, char).**

Исходные данные (строки, символы) вводить к клавиатуры, результат (строки, символы) выводить на экран дисплея. При разработке интерфейса использовать компоненты **CheckBox, RadioButton, CheckedListBox, GroupBox, TextBox, Label.**

Задание 2. Разработать программу, которая позволяет заменить слово в тексте, выбранным по его номеру, на новое слова. Исходный текст вводится с клавиатуры.

При разработке интерфейса приложения использовать компоненты **Label, Button, TextBox, NumericUpDown** (один из вариантов интерфейса представлен на рис. 1).

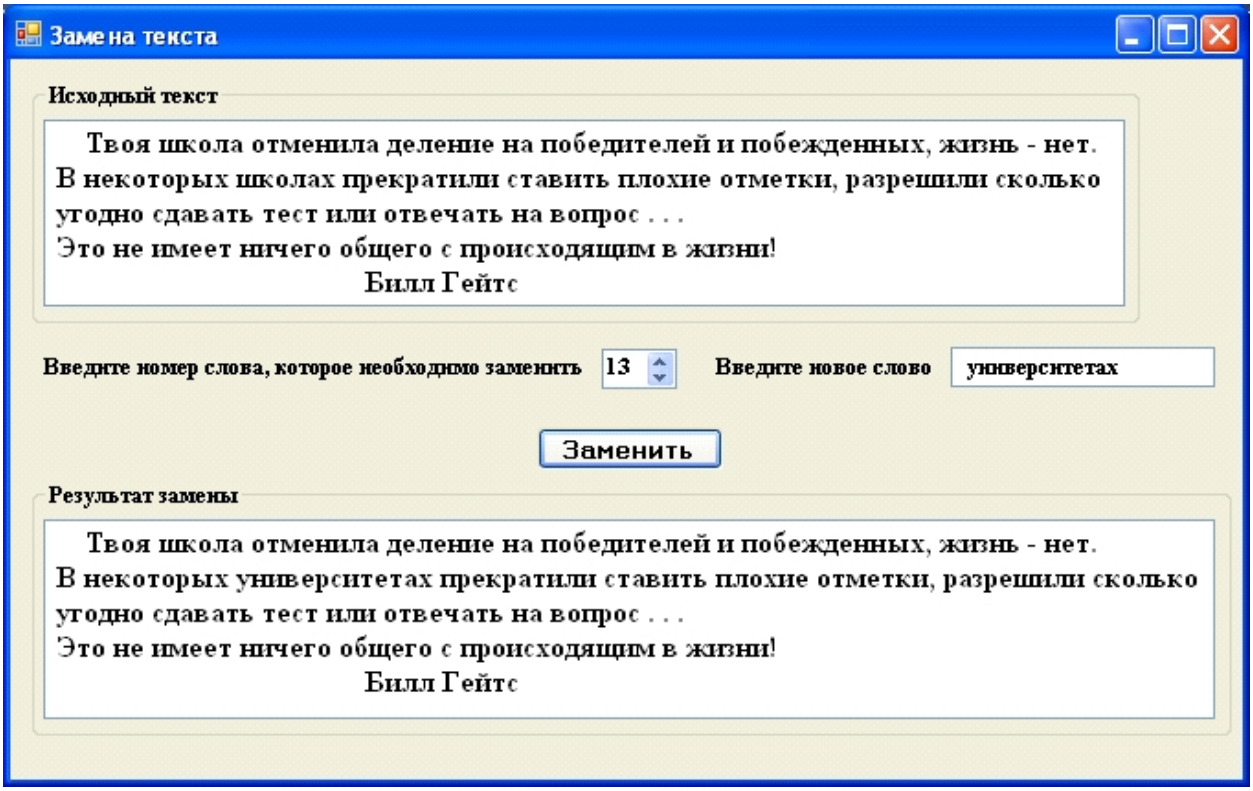

Рис. 1. Внешний вид приложения «Замена текста»

Задание 3. Разработать программу, которая позволяет осуществлять замену слов в тексте на маскирующие символы. Исходный текст вводится с клавиатуры.

При разработке интерфейса приложения использовать компоненты **Label, Button, TextBox, ListBox** (один из вариантов интерфейса представлен на рис. 2).

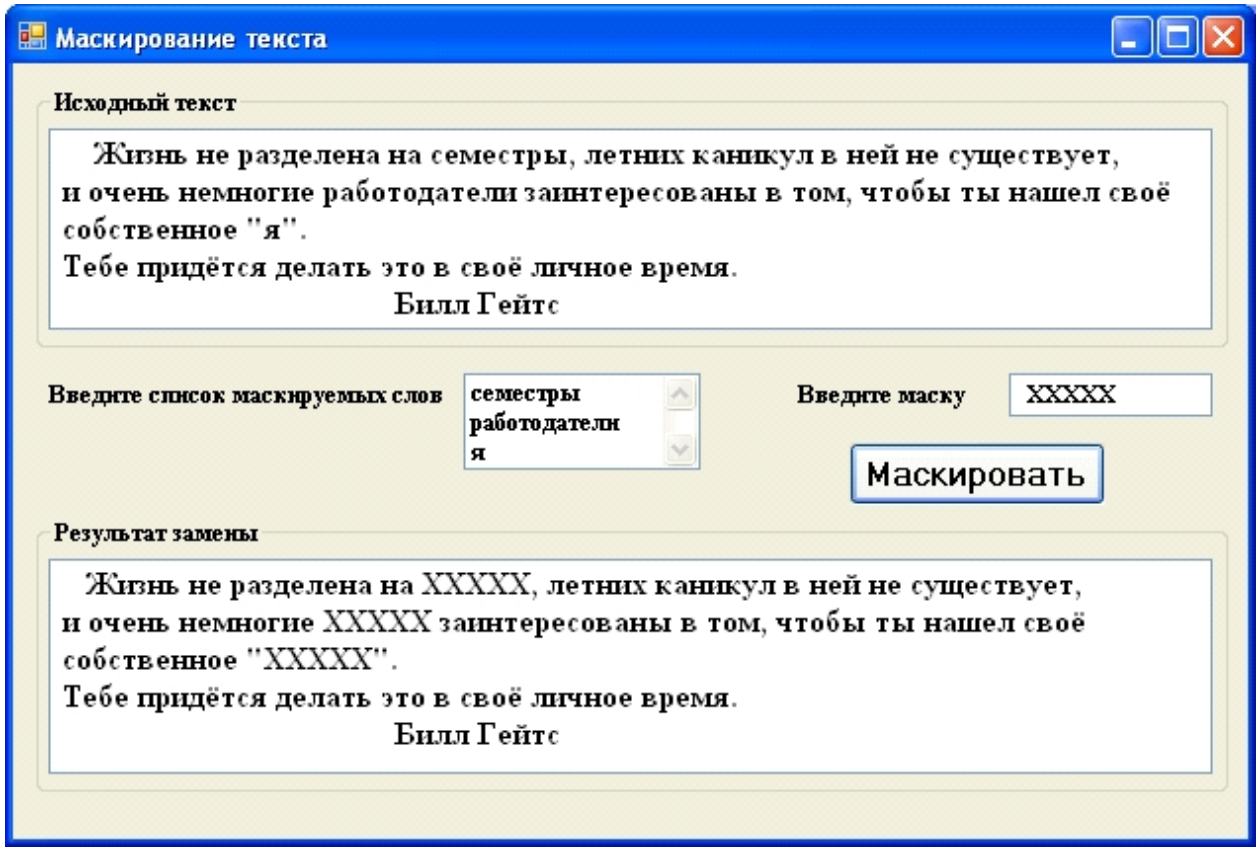

Рис. 2. Внешний вид приложения «Маскирование текста»

3. Контрольные вопросы

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Культин Н.Б. Microsoft Visual C# в задачах и примерах. – СПб.: БХВ-Петербург, 2009. – 320 с.: ил.
3. Лабор В.В. Си Шарп: Создание приложений для Windows. – Мн.: Харвест, 2003. – 384 с.
4. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
5. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
6. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
7. Фаронов В.В. Программирование на языке C#. – СПб.: Питер, 2007. – 240 с.: ил.
8. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011. – 560с.: ил.