МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Брянский государственный технический университет

Утверждаю
Ректор университета
_____О.Н. Федонин
«_____»_____2013 г.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

РАБОТА СО СТРОКАМИ

Методические указания
к выполнению лабораторной работы
для студентов очной формы обучения
специальностей 090303 – «Информационная безопасность автоматизированных систем»,
090900 – «Информационная безопасность»

Брянск 2013

УДК 004.43

Языки программирования. Работа со строками: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 20 с.

Разработали:
Ю.А. Леонов, к.т.н., доц.,
Е.А. Леонов, к.т.н., доц.

Научный редактор: Ю.М. Казаков
Редактор издательства: Л.И. Афонина
Компьютерный набор: Ю.А. Леонов

Рекомендовано кафедрой «Компьютерные технологии и системы» БГТУ (протокол № 2 от 24.10.12)

Темплан 2013 г., п.

Подписано в печать Формат 60х84 1/16. Бумага офсетная.
Офсетная печать.
Усл. печ. л. 1,16 Уч. – изд. л. 1,16 Тираж 20 экз. Заказ Бесплатно

Издательство брянского государственного технического университета,
241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Харьковская, 9

1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение основ работы со строками и составление алгоритмов с их использованием.

Продолжительность работы – 2 ч.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Часто при решении задач необходимо использовать данные представленные в виде набора символов называемые *строками*.

В языке программирования C# существуем несколько классов предназначенных для работы со строками, среди них можно выделить класс *String* (псевдоним *string*) и класс *StringBuilder*. Рассмотрим особенности работы с этими классами.

2.1. Работа со строками класса *String*

Класс *String* находится в пространстве имен *System* (*System.String*) и представляет последовательность из нуля или более символов в кодировке Юникод. Класс *string* – это псевдоним для класса *String* платформы *.NET Framework*.

Класс *String* имеет набор полей, свойств и методов по работе со строками. При использовании данного класса исходная строка не изменяется, а можно лишь получать требуемый результат в выходных данных используемого метода.

Инициализация строки производится следующим образом:

string <имя_строки> = “значение строки”;

Каждый символ в строке неявным образом пронумерован начиная с нуля, и доступ к символу строки осуществляется таким же образом, как и элементу массива, т.е. необходимо указать имя строковой переменной и в квадратных скобках указать номер символа. Например, для строки *s* присвоим значение «строка» и осуществим вывод на экран символ строки с индексом 2:

```
string s = “строка”;
Console.WriteLine(s[2]);
```

В результате на экране отобразится символ «р».

Для определения количества символов в строке необходимо использовать свойство *Length*. Конкатенацию строк (присоединение или слияние строк) можно выполнить при помощи операции «+».

Для работы со строками приведем некоторые из основных методов класса *string* (табл. 1).

Таблица 1

Основные методы для работы со строками *string*

Название	Назначение
CompareTo	Сравнивает строку, для которой вызван данный метод с переданной строкой *b*, и показывает относительное положение строк в алфавитном порядке.
CopyTo	Копирует заданное количество символов строки с указанной позиции источника (*source*) в массив символов (*char[]*) с указанной позиции.
IndexOf	Находит первое вхождение заданной подстроки в строке.
Insert	Возвращает новую строку, в которую вставлена переданная подстрока в указанную позицию строки.
PadLeft, PadRight	Возвращает новую строку полученную добавлением символов до заданной длины повторяющимися символами слева (*PadLeft*) или справа (*PadRight*).
Remove	Возвращает новую строку, из которой удалена указанная подстрока.
Replace	Возвращает новую строку, в которой вхождения переданной подстроки заменены на другую указанную подстроку.
Split	Возвращает строковый массив полученный разбиением строки на подстроки, используя в качестве разделителя массив символов.
Substring	Возвращает подстроку, извлеченную из строки с указанного индекса установленной длины.
ToUpper, ToLower	Возвращает новую строку, в которой символы переведены в верхний регистр (*ToUpper*) или в нижний регистр (*ToLower*).
Trim	Возвращает новую строку, в которой удалены все начальные и конечные символы пробела.

Примеры использования методов

Почти каждый из представленных методов имеет множество вариантов реализации (перегрузок). Рассмотрим некоторые из реализаций методов (табл. 1).

CompareTo

public int CompareTo(string strB)

```
string a = “строка”;
int v = a.CompareTo("строка"); // v == 0 – строки равны
```

int v = a.CompareTo("ястрока"); // v < 0 – строка *a* окажется выше чем переданная подстрока «ястрока» в порядке сортировки

int v = a.CompareTo("астрока"); // v > 0 – строка *a* окажется ниже чем переданная подстрока «астрока» в порядке сортировки

CopyTo

public void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count)

```
string a = "0123456789";
char[] c = new char[6] {'с','т','р','о','к','а'};
a.CopyTo(1, c, 3, 2); // c = «стр12а»
```

IndexOf

public int IndexOf(string value)

```
string b="строка1строка2";
int v = b.IndexOf("рок"); // v = 2
```

Insert

public string Insert(int startIndex, string value)

```
string a = "строка", b;
b = a.Insert(2, "123"); // b = «ст123рока»
```

PadLeft

public string PadLeft(int totalWidth)

public string PadLeft(int totalWidth, char paddingChar)

```
string a = "строка", b;
b = a.PadLeft(8); // b = «  строка»
b = a.PadLeft(8, ‘+’); // b = «++строка»
```

Remove

public string Remove(int startIndex)

public string Remove(int startIndex, int count)

```
string a = "строка", b;
```

```
b = a.Remove(3); // b = «стр»
b = a.Remove(1, 3); // b = «ска»
```

Replace

public string Replace(string oldValue, string newValue)

```
string a = "строка", b;
b = a.Replace("тро", "вал"); // b = «свалка»
```

Split

public string[] Split(params char[] separator)

```
string a = "23,10.2012";
string[] b = a.Split(new char[] {'.', ','}); // b = {“23”, “10”, “2012”}
```

Substring

public string Substring(int startIndex)

public string Substring(int startIndex, int length)

```
string a = "строка", b;
b = a.Substring(2); // b = «рока»
b = a.Substring(2, 3); // b = «рок»
```

ToUpper

public string ToUpper()

```
string a = "строка", b;
b = a.ToUpper(); // b = «СТРОКА»
```

Trim

public string Trim()

```
string a = " стро ка  ";
string b = a.Trim(); // b = «стро ка»
```

2.2. Работа со строками класса *StringBuilder*

Класс *StringBuilder* находится в пространстве имен *System.Text* (*System.Text.StringBuilder*). При работе со строками методы класса *String* не изменяют строку, а лишь возвращают измененную копию строки. Если необходимо изменять строки без создания копий, то можно воспользоваться классом *StringBuilder*.

Для создания строки нужно воспользоваться одним из конструкторов класса *StringBuilder*. Представим некоторые из них:

StringBuilder(string value);

StringBuilder(int capacity);

StringBuilder(string value, int capacity);

Параметр *value* (значение) является значением строки, а параметр *capacity* (емкость) является начальным размером выделяемой памяти устанавливаемой в количестве символов. В случае, если начального размера не хватает, то выделяется дополнительная память.

Инициализация строки производится следующим образом:

StringBuilder <имя_строки> = new <конструктор>;

Например, инициализируем строку со значением «строка»:

StringBuilder s = new StringBuilder(“строка”);

или создадим строку начальным размером 100 символов:

StringBuilder s = new StringBuilder(100);

Приведем основные методы класса *StringBuilder* (табл. 2).

Таблица 2

Основные методы для работы со строками *StringBuilder*

Название	Назначение
Append	Добавляет строку к текущей строке
AppendFormat	Добавляет строку, сформированную в соответствии со спецификатором формата
Clear	Удаляет все символы из текущей строки
CopyTo	Копирует символы из текущей строки в указанный символьный массив
Insert	Вставляет подстроку в строку
Remove	Удаляет символ из текущей строки
Replace	Заменяет все вхождения символа другим символом или вхождения подстроки другой подстрокой
ToString	Возвращает текущую строку в виде объекта *string*

Кроме методов в классе присутствуют свойства: *Capacity*, *MaxCapacity*, *Length*.

Capacity – возвращает или задает максимальное число символов, которое может содержаться в памяти назначенной текущим экземпляром.

MaxCapacity – возвращает максимальную емкость данного экземпляра.

Length – возвращает или задает длину текущего экземпляра.

Рассмотрим работу представленных методов (табл. 2).

Примеры использования методов

Также как и для методов класса *string* для методов класса *StringBuilder* существуют множество перегрузок. Рассмотрим некоторые варианты реализаций методов. Методы: *CopyTo, Insert, Remove, Replace* рассматриваться не будут, т.к. они работают аналогично методом класса *string*.

Append

public StringBuilder Append(string value)

```
StringBuilder a = new StringBuilder("строка1");
StringBuilder b = new StringBuilder();
b = a.Append("строка2"); // b = «строка1строка2»
```

AppendFormat

public StringBuilder AppendFormat(string format,params object[] args)

```
StringBuilder a = new StringBuilder("Умножение чисел: ");
StringBuilder b = new StringBuilder();
int dig1 = 5 * 2; double dig2 = 2.3 * 5.7;
b = a.AppendFormat("5 * 2 = {0}, 2.3 * 5.7 = {1}", dig1, dig2); // b = «Умножение чисел: 5 * 2 = 10, 2.3 * 5.7 = 13,11»
```

Clear

public StringBuilder Clear()

```
StringBuilder a = new StringBuilder("строка");
StringBuilder b = new StringBuilder();
b = a.Clear(); // b = «»
```

ToString

public string ToString()

```
StringBuilder a = new StringBuilder("строка");
string b;
b = a.ToString(); // b = «строка»
```

2.3. Управляющие последовательности и вывод служебных символов

В C# строки могут содержать управляющие последовательности (табл. 3).

Таблица 3

Основные управляющие последовательности

Управляющие последовательности	Назначение
\’	Вставить одинарную кавычку в строку
\”	Вставить двойную кавычку в строку
\\	Вставить в строку обратный слэш.
\b	Вернуться на одну позицию
\n	Вставить новую строку
\r	Вставить возврат каретки
\t	Вставить горизонтальный символ табуляции
\v	Вставить вертикальный символ табуляции

Например, запишем с помощью управляющей последовательности путь к файлу: *string path = "C:\\Temp\\temp.dat";*

Помимо управляющих последовательностей, в C# предусмотрен специальный префикс @ для дословного вывода строк вне зависимости от наличия в них управляющих символов. Для предыдущего примера путь к файлу запишется так: *string path = @"C:\Temp\temp.dat";*

2.4. Составное форматирование строк

В качестве входных данных для составного форматирования в *.NET Framework* используется список объектов и строка составного формата. Строка составного формата состоит из фиксированного текста, в который включены индексированные местозаполнители, которые называются элементами форматирования и соответствуют объектам из списка. Операция форматирования создает результирующую строку, состоящую из исходного фиксированного текста, в который включено строковое представление объектов из списка.

Составное форматирование поддерживается такими методами, как *Format* и *AppendFormat*, а также некоторыми перегрузками методов *WriteLine* и *TextWriter.WriteLine*. Метод *String.Format* возвращает отформатированную результирующую строку, метод *AppendFormat* добавляет отформатированную результирующую строку к объекту *StringBuilder*, метод *Console.WriteLine* выводит отформатированную результирующую строку на консоль, а метод *TextWriter.WriteLine* записывает отформатированную результирующую строку в поток или файл.

Строка составного формата

Строка составного формата и список объектов используются в качестве аргументов методов, поддерживающих составное форматирование. Строка составного формата состоит из блоков фиксированного текста числом от нуля и больше, перемежаемых одним или несколькими элементами форматирования. Фиксированным текстом может являться произвольная строка, а каждый элемент форматирования должен соответствовать объекту или упакованной структуре из списка. В ходе составного форматирования создается новая результирующая строка, в которой все элементы форматирования заменены на строковое представление соответствующих объектов из списка.

Например, используем составное форматирование для вывода целого и вещественного значений на экран:

```
Console.WriteLine("Целое число: {0}, вещественное число: {1}", 5, 10.1);
```

Фиксированным текстом в данном примере является «Целое число: » и «, вещественное число: ». Элементами форматирования являются «{0}» и «{1}», которые определяют места для вывода целого и вещественного чисел. На экране будет следующее: «Целое число: 5, вещественное число: 10,1».

Синтаксис элементов форматирования

Каждый элемент форматирования имеет следующий синтаксис описания:

{индекс [, выравнивание][: строкаФормата]}

Фигурные скобки обязательно указывать, а то, что указано в квадратных скобках может быть опущено.

Обязательный компонент ***индекс*** – это число, определяющее соответствующий объект из списка. Индексация элементов ведется от нуля. Иными словами, элемент форматирования с индексом 0 отвечает за формат первого объекта в списке, элемент форматирования с индексом 1 служит для форматирования второго объекта в списке и т. д.

На один и тот же элемент в списке объектов может ссылаться сразу несколько элементов форматирования – достигается это путем задания одинакового описателя параметра. Например, одно и тоже числовое значение можно отформатировать в шестнадцатеричном, экспоненциальном и десятичном виде путем задания следующей строки составного форматирования: «{0:X} {0:E} {0:N}».

Любой элемент форматирования может ссылаться на произвольный объект списка. Например, если имеется три объекта, то можно отформатировать сначала второй, а затем первый и третий объекты, задав следующую строку составного форматирования: "{1} {0} {2}". Объекты, на которые не ссылаются элементы форматирования, пропускаются. Если описатель параметра ссылается на элемент за пределами списка объектов, то во время выполнения создается исключение.

Необязательный компонент ***выравнивание*** – это целое число со знаком, которое служит для указания желательной ширины поля форматирования. Если значение *выравнивание* меньше длины форматируемой строки, то *выравнивание* пропускается, и в качестве значения ширины поля используется длина форматируемой строки. Форматируемые данные выравниваются в поле по правому краю, если *выравнивание* имеет положительное значение, или по левому краю, если *выравнивание* имеет отрицательное значение. При необходимости отформатированная строка дополняется пробелами. При использовании компонента *выравнивание* необходимо поставить запятую.

Необязательный компонент ***строкаФормата*** – это строка формата, соответствующая типу форматируемого объекта. Например, если форматируемый объект является объектом *DateTime*, то используется строка стандартного или настраиваемого формата даты и времени, а если объект является значением числового типа (sbyte, byte, short, ushort, int, uint, long, ulong), то используется строка стандартного или настраиваемого числового формата. При использовании компонента *строкаФормата* необходимо поставить двоеточие.

Наиболее часто требуется отформатировать вывод целых и вещественных чисел. Рассмотрим стандартное и настраиваемое форматирование числовых данных.

Строки стандартных числовых форматов

Строки стандартных числовых форматов служат для форматирования стандартных числовых типов. Стандартная строка числового формата имеет вид *Axx*, где *A* является символом буквы, называемой *спецификатором формата*, а *xx* является опциональным целым числом, называемым *спецификатором точности*. Спецификатор точности находится в диапазоне от 0 до 99 и влияет на число цифр в результате. Любая строка числового формата, содержащая более одной

буквы, включая пробелы, интерпретируется как строка пользовательского числового формата.

Строки стандартного числового формата поддерживаются некоторыми перегрузками метода *ToString* всех числовых типов (классов). Например, можно задать строку числового формата для методов *ToString(string format)* и *ToString(string format, IFormatProvider provider)* класса *int*.

В табл. 4 описаны спецификаторы стандартных числовых форматов и их названия, а также описания спецификаторов, в которых представлены результаты работы спецификаторов и указаны поддерживаемые типы данных.

Таблица 4

Спецификаторы стандартных числовых форматов

Спецификатор	**Имя**	**Описание**
"C" или "c"	Валюта	*Результат:* значение валюты. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* количество цифр дробной части.
"D" или "d"	Десятичный	*Результат:* целочисленные цифры с необязательным отрицательным знаком. *Поддерживается:* только целочисленными типами данных. *Спецификатор точности:* минимальное число цифр.
"E" или "e"	Экспоненциальный	*Результат:* экспоненциальная нотация. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* количество цифр дробной части.
"F" или "f"	Фиксированная запятая	*Результат:* цифры целой и дробной частей с необязательным отрицательным знаком. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* количество цифр дробной части.

Окончание табл. 4

Специ-фикатор	Имя	Описание
"G" или "g"	Общие	*Результат:* наиболее компактная запись из двух вариантов: экспоненциального и с фиксированной запятой. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* количество значащих цифр.
"N" или "n"	Числовой	*Результат:* цифры целой и дробной частей, разделители групп и разделитель целой и дробной частей с необязательным отрицательным знаком. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* желаемое число знаков дробной части.
"P" или "p"	Процент	*Результат:* число, умноженное на 100 и отображаемое с символом процента. *Поддерживается:* всеми числовыми типами данных. *Спецификатор точности:* желаемое число знаков дробной части.
"R" или "r"	Туда-обратно	*Результат:* строка, дающая при обратном преобразовании идентичное число. *Поддерживается:* Single, Double и BigInteger. *Спецификатор точности:* игнорируется.
"X" или "x"	Шестнадцатеричный	*Результат:* шестнадцатеричная строка. *Поддерживается:* только целочисленными типами данных. *Спецификатор точности:* число цифр в результирующей строке.

Представим примеры использования спецификаторов в порядке их перечисления в табл. 4.

Примеры использования спецификаторов стандартных числовых форматов

Представленные примеры следует рассматривать в порядке написания слева направо, сверху вниз, т.е. для переменных всегда актуально последнее значение.

```
decimal d = 10.2355m; string s;
```

```
// Метод CreateSpecificCulture в классе CultureInfo устанавливает регион, для которого отображается число, по умолчанию берется установленный регион из операционной системы
s = d.ToString("c2", CultureInfo.CreateSpecificCulture("en-US")); // s = «$10.24»
s = d.ToString("c2", CultureInfo.CreateSpecificCulture("ru")); // s = «10,24 р.»
s = d.ToString("c2"); // s = «10,24 р.», по умолчанию установлен регион «Россия»
int i = 5;
s = i.ToString("d"); // s = «5»
s = i.ToString("d4"); // s = «0005»
s = d.ToString("e3"); // s = «1,024e+001»
d = 7.23m;
s = d.ToString("f1") // s = «7,2»
s = d.ToString("f4") // s = «7,2300»
s = d.ToString("g2") // s = «7,2»
s = d.ToString("g4") // s = «7,23»
d = 12345.12345m;
s = d.ToString("n2") // s = «12 345,12»
d = 1m;
s = d.ToString("p") // s = «100,00%»
d = 0.151m;
s = d.ToString("p") // s = «15,10%»
double d1 = 123456789.12345678;
s = d1.ToString("r") // s = «123456789,12345678»
i = 15;
s = i.ToString("x") // s = «f»
i = 255;
s = i.ToString("X") // s = «FF»
```

В случае, если стандартные форматы не подходят для вывода чисел можно использовать строки настраиваемых числовых форматов.

Строки настраиваемых числовых форматов

Строки настраиваемых числовых форматов поддерживаются теми же методами что и строки стандартных числовых форматов.

В табл. 5 описаны спецификаторы настраиваемых числовых форматов и результаты их работы.

Таблица 5

Спецификаторы настраиваемых числовых форматов

Спецификатор	Имя	Описание
"0"	Знак-заместитель нуля	Заменяет ноль соответствующей цифрой, если такая имеется. В противном случае в результирующей строке будет стоять ноль.
"#"	Заместитель цифры	Заменяет знак "#" соответствующей цифрой, если такая имеется. В противном случае в результирующей строке цифра стоять не будет.
"."	Разделитель	Определяет расположение разделителя целой и дробной частей в результирующей строке.
","	Разделитель групп и масштабирование чисел	Служит в качестве спецификатора разделителя групп и спецификатора масштабирования чисел. В качестве разделителя групп вставляет локализованный символ-разделитель групп между всеми группами. В качестве спецификатора масштабирования чисел делит число на 1000 для всех указанных запятых.
"%"	Заместитель процентов	Умножает число на 100 и вставляет локализованный символ процента в результирующую строку.
"‰"	Местозаполнитель промилле	Умножает число на 1000 и вставляет локализованный символ промилле в результирующую строку.
"E0" "E+0" "E-0" "e0" "e+0" "e-0"	Экспоненциальная нотация	Если за этим спецификатором следует по меньшей мере один ноль (0), результат форматируется с использованием экспоненциальной нотации. Регистр ("E" или "e") определяет регистр символа экспоненты в результирующей строке. Минимальное число цифр экспоненты определяется количеством нулей, стоящих за символом "E" или "e". Знак "+" указывает на то, что перед экспонентой всегда должен ставиться символ знака. Знак "-" указывает на то, что символ знака должен ставиться только в случае, если экспонента имеет отрицательное значение.
\	Escape-символ	Указывает на то, что следующий за ним символ должен рассматриваться как литерал, а не как спецификатор настраиваемого формата.
'строка' "строка"	Разделитель строк-литералов	Указывает на то, что заключенные в разделители символы должны быть скопированы в результирующую строку без изменений.

Окончание табл. 5

Специ-фикатор	Имя	Описание
;	Разделитель секций	Определяет секции с раздельными строками формата для положительных чисел, отрицательных чисел и нуля.
Другой	Все остальные символы	Символ копируется в результирующую строку без изменений.

Представим примеры использования спецификаторов в порядке их перечисления в табл. 5.

Примеры использования спецификаторов настраиваемых числовых форматов

Представленные примеры следует рассматривать в порядке написания слева направо, сверху вниз, т.е. для переменных всегда актуально последнее значение.

```
decimal d = 10.1234m; string s;
s = d.ToString("0.00"); // s = «10,12»
d = 0.456m;
s = d.ToString("#.##"); // s = «,46»
d = 1234567m;
s = d.ToString("##,##"); // s = «1 234 567»
d = 0.12345m;
s = d.ToString("##.00 %"); // s = «12,35 %»
d = 0.02345m;
s = d.ToString("#0.00 " + '\u2030'); // s = «23,45 ‰»
d = 2103.5612m;
s = d.ToString("0.0##e+00"); // s = «2,104e+03»
int i = 12345;
s = i.ToString("\\###00\\%"); // s = «#12345%»
i = 15;
s = i.ToString("#' градусов'"); // s = «15 градусов»
d = -12.345m;
s = d.ToString("#0.0#;(#0.0#)"); // s = «(12,35)», т.к. число отрицательное, то число форматируется по второй секции
i = 5;
s = i.ToString("# @"); // s = «5 @»
```

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения лабораторной работы необходимо предварительно ознакомиться с теоретической частью в случае необходимости требуется использовать дополнительный учебно-методический материал. После ознакомления с теорией требуется ознакомиться с заданием (см. список заданий) и составить алгоритм решения задачи. Далее требуется на языке C# написать программу, реализующую составленный алгоритм.

4. СПИСОК ЗАДАНИЙ

1. Необходимо зашифровать введенную с клавиатуры строку, поменяв местами первый символ с последним, второй с предпоследним и т.д.
2. Найти и заменить в строке определенный символ, введенный с клавиатуры. Программа должна спрашивать заменяемый и замещающий символ, а также выводить его номер позиции в строке.
3. Имеется массив слов. Необходимо создать процедуру, которая ищет в массиве, переданное в параметре процедуры, слово и выводит набор индексов массива совпадающих элементов.
4. Необходимо найти все гласные буквы во введенном пользователем слове и осуществить вывод на экран этого слова с помеченными цветом гласными буквами.
5. Определить и вывести на экран номера позиций и количество повторений запрашиваемой подстроки в строке, введенной с клавиатуры.
6. Определить количество слов в предложении, при условии, что каждое слово отделяется от другого пробелом.
7. Необходимо отсортировать по возрастанию слова находящиеся в массиве.
8. Необходимо вывести из массива только те строки, в которых количество гласных букв превышает количество согласных букв.
9. Определить самое длинное и самое короткое слово в предложении, при условии, что каждое слово отделяется от другого пробелом.
10. Определить самое длинное и самое короткое предложение, при условии, что предложения отделяются друг от друга с помощью определенных в русском языке знаков пунктуации.

11. Создать метод, который считает сумму цифр переданных в строковой переменной. Все цифры имеют один разряд. Если в строковой переменной встречаются буквы их необходимо пропустить.
12. Написать программу, которая распознает буквенные обозначения аккордов, переданных в строковой переменной. Определяется только минор, мажор, септаккорд; пример: Am^7. Формат вывода результата: Септ аккорд в Ля минор;
13. Создать метод, который вырезает из строковой переменной цифры.
14. Создать метод, который сравнивает две даты, переданные в строковой переменной, и результат сравнения выдает в имени функции.
15. Создать метод, который проверяет корректность введенной даты. Формат даты передается в строковой переменной (маска), где *d* – день, *m* – месяц, *y* – год, количество этих букв говорит о количестве позиций дня, месяца или года. Например: dd.mm.yyyy.
16. Создать метод, который увеличивает или уменьшает дату на определенное количество дней (передается во входном параметре), результатом работы функции будет «новая» дата.
17. Создать метод, который разбирает дату, переданную в строковой переменной в целочисленные переменные числа, месяца и года. Формат даты передается в строковой переменной (маска), где *d* – день, *m* – месяц, *y* – год, количество этих букв говорит о количестве позиций дня, месяца или года. Например: *dd.mm.yyyy*.
18. Создать метод, который увеличивает или уменьшает определенное количество месяцев к переданной в строковой переменной дате, результирующая дата возвращается в строковой переменной.
19. Создать метод, который подсчитывает количество дней прошедших от начала года в переданной дате.
20. Создать метод, который подсчитывает количество гласных, согласных букв и количество символов относящихся к цифрам.
21. Дана строка. Вывести подстроку, расположенную между первым и вторым пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку.
22. Дана строка, содержащая полное имя файла. Необходимо выделить из этой строки имя файла с расширением и путь к файлу.

23. Дана строка, состоящая из русских слов. Вывести строку, которая содержит эти же слова, расположенные в обратном порядке.
24. Дана строка, состоящая из английских слов. Вывести строку, содержащую эти же слова, расположенные в алфавитном порядке.
25. Дана строка на русском языке. Зашифровать ее выполнив циклическую замену каждой буквы на следующую за ней в алфавите и сохранив при этом регистр букв («А» перейдет в «Б», «Б» в «В», «Я» в «А»). Букву «Ё» в алфавите не учитывать.
26. Дана строка, состоящая из русских слов, набранных заглавными буквами и разделенных пробелами. Найти количество слов, которые начинаются и заканчиваются одной и той же буквой.
27. Дана строка, состоящая из английских слов, набранных заглавными буквами и разделенных пробелами. Найти количество слов, которые содержат хотя бы одну букву «А».
28. Дана строка, состоящая из русских слов и разделенных пробелами. Найти длину самого короткого слова.
29. Дана строка, состоящая из английских слов, набранных заглавными буквами и разделенных пробелами. Преобразовать каждое слово в строке, заменив в нем все последующие вхождения его первой буквы на символ «.». Например, слово «МИНИМУМ» надо преобразовать в «МИНИ.У.»
30. Дана строка, содержащая полное имя файла. Выделить из этой строки название первого каталога (без символов «\»). Если файл содержится в корневом каталоге, то вывести символ «\».

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие способы представления строк в C# вы знаете?
2. Каким образом можно определить размер строки?
3. Какие существуют способы сравнения строк?
4. Какие методы предназначены для работы с классами *string* и *StringBuilder*?
5. Каким образом инициализируется строки представленные классами *string* и *StringBuilder*?
6. Чем принципиально различаются строки представленные классами *string* и *StringBuilder*?
7. Каким образом можно преобразовать численные данные в строковые и наоборот?
8. Что называют составным форматированием строк?

9. Какие вы знаете спецификаторы стандартных и настраиваемых числовых форматов?

6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная

1. Павловская, Т. А. C#. Программирование на языке высокого уровня. – Изд.: Питер, 2009. – 432с.
2. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4. – Изд.: Вильямс, 2011. – 1392с.
3. Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер. C# 4.0 и платформа .NET 4 для профессионалов. – Изд.: Питер, 2011. – 1440с.

Дополнительная

4. Тыртышников, Е.Е. Методы численного анализа. Учебное пособие. – Изд.: МГУ, 2006. – 281с.
5. Джесс Либерти. Программирование на C#. – Изд.: КноРус, 2003. – 688с.
6. Харви Дейтел. C# в подлиннике. Наиболее полное руководство. – Изд.: БХВ-Петербург, 2006. – 1056с.