**Лекция 12. Функции работы со строками**

**Цель лекции:** Формирование у студентов знаний, связанных с использованием функций работы со строками.

В результате изучения данной лекции студенты будут способны:

* Продемонстрировать знание синтаксиса функций обработки строк
* Различать статические и нестатические методы обработки строк

**Методы класса String**

Класс типа string содержит ряд методов для обращения со строками. Некоторые методы принимают параметр типа StringComparison. Это перечислимый тип, определяющий различные значения, которые определяют порядок сравнения символьных строк. Нетрудно догадаться, что символьные строки можно сравнивать разными способами. Например, их можно сравнивать на основании двоичных значений символов, из которых они состоят. Такое сравнение называется *порядковым.* Строки можно также сравнивать с учетом различных особенностей культурной среды, например, в лексикографическом порядке. Это так называемое сравненение *с учетом культурной qpedbi.* (Учитывать культурную среду особенно важно в локализуемых приложениях.) Кроме того, строки можно сравнивать *с учетом* или *без учета* регистра. Несмотря на то что существуют перегружаемые варианты методов Compare(), Equals(), IndexOf() и LastIndexOf(), обеспечивающие используемый по умолчанию подход к сравнению символьных строк, в настоящее время считается более приемлемым явно указывать способ требуемого сравнения, чтобы избежать неоднозначности, а также упростить локализацию приложений. Именно поэтому здесь рассматривают разные способы сравнения символьных строк.

Как правило и за рядом исключений, для сравнения символьных строк с учетом культурной среды (т.е. языковых и региональных стандартов) применяется способ StringComparison.CurrentCulture. Если же требуется сравнить строки только на основании значений их символов, то лучше воспользоваться способом StringComparison.Ordinal, а для сравнения строк без учета регистра — одним из двух способов: StringComparison. CurrentCulturelgnoreCase или StringComparison.OrdinallgnoreCase.

Метод Compare () объявляется как static. Подробнее о модификаторе static речь пойдет в следующих лекциях, а до тех пор вкратце поясним, что он обозначает следующее: метод Compare () вызывается по имени своего класса, а не по его экземпляру. Следовательно, для вызова метода Compare () служит следующая общая форма:

*результат =* string.Compare( *strl* , *str2, способ);*

где *способ* обозначает конкретный подход к сравнению символьных строк.

Обратите также внимание на методы ToUpper() и ToLower(), преобразующие содержимое строки в символы верхнего и нижнего регистра соответственно. Их формы содержат параметр CultureInfо, относящийся к классу, в котором описываются атрибуты культурной среды, применяемые для сравнения. В примерах используются текущие настройки культурной среды (т.е. текущие языковые и региональные стандарты). Эти настройки указываются при передаче методу аргумента Culturelnfо. CurrentCulture. Класс Culturelnfо относится к пространству имен System.Globalization. Любопытно, имеются варианты рассматриваемых здесь методов, в которых текущая культурная среда используется по умолчанию, но во избежание неоднозначности в примерах из этой книги аргумент Culturelnfо.CurrentCulture указывается явно.

Объекты типа string содержат также свойство Length, где хранится длина строки.

Отдельный символ выбирается из строки с помощью индекса, как в приведенном ниже фрагменте кода.

string str = "тест";

Console.WriteLine(str [0] ) ;

В этом фрагменте кода выводится символ "т", который является первым в строке "тест". Как и в массивах, индексирование строк начинается с нуля. Следует, однако, иметь в виду, что с помощью индекса нельзя присвоить новое значение символу в строке. Индекс может служить только для выборки символа из строки.

Для проверки двух строк на равенство служит оператор ==. Как правило, если оператор == применяется к ссылкам на объект, то он определяет, являются ли они ссылками на один и тот же объект. Совсем иначе обстоит дело с объектами типа string. Когда оператор == применяется к ссылкам на две строки, он сравнивает содержимое этих строк. Это же относится и к оператору ! =. В обоих случаях выполняется порядковое сравнение. Для проверки двух строк на равенство с учетом культурной среды служит метод Equals (), где непременно нужно указать способ сравнения в виде аргумента StringComparison . CurrentCulture. Следует также иметь в виду, что метод Compare () служит для сравнения строк с целью определить отношение порядка, например для сортировки. Если же требуется проверить символьные строки на равенство, то для этой цели лучше воспользоваться методом Equals () или строковыми операторами.

В приведенном ниже примере программы демонстрируется несколько операций со строками.

// Некоторые операции со строками.

using System;

using System.Globalization;

class StrOps {

static void Main() {

string strl    =    "Программировать    в    .NET    лучше    всего    на    С#.";

string str2    =    "Программировать    в    .NET    лучше    всего    на    С#.";

string str3    =    "Строки в C# весьма эффективны.";

string strUp,    strLow;

int result, idx;

Console.WriteLine("strl: " + strl);

Console.WriteLine("Длина строки strl: " + strl.Length);

// Создать варианты строки strl, набранные // прописными и строчными буквами.

strLow = strl.ToLower(Cirlturelnfo.CurrentCulture);

strUp = strl.ToUpper (Culturelnfo.CurrentCulture);

Console.WriteLine("Вариант строки strl, " +

"набранный строчными буквами:\n " + strLow); Console.WriteLine("Вариант строки strl, " +

"набранный прописными буквами:\n " + strUp);

Console.WriteLine();

// Вывести строку strl посимвольно.

Console.WriteLine("Вывод строки strl посимвольно.") ; for (int i=0; i < strl.Length; i++)

Console.Write(strl[i]);

Console.WriteLine("\n");

// Сравнить строки способом порядкового сравнения, if (strl == str2)

Console.WriteLine("strl == str2"); else

Console.WriteLine("strl != str2"); if (strl == str3)

Console.WriteLine("strl == str3"); else

Console.WriteLine("strl != str3");

// Сравнить строки с учетом культурной среды.

result = string.Compare(str3, strl, StringComparison.CurrentCulture) ; if(result == 0)

Console.WriteLine("Строки strl и str3 равны"); else if (result < 0)

Console.WriteLine("Строка strl-меньше строки str3"); else

Console.WriteLine("Строка strl больше строки str3");

Console.WriteLine();

// Присвоить новую строку переменной str2. str2 = "Один Два Три Один";

// Поиск подстроки.

idx = str2.IndexOf("Один", StringComparison.Ordinal);

Console.WriteLine("Индекс первого вхождения подстроки <Один>: " + idx)

idx = str2.LastlndexOf("Один", StringComparison.Ordinal);

Console.WriteLine("Индекс последнего вхождения подстроки <0дин>: " + idx) ;

}

}

При выполнении этой программы получается следующий результат.

strl: Программировать в .NET лучше всего на С#.

Длина строки strl: 41

Вариант строки strl, набранный строчными буквами: программировать в .net лучше всего на с#.

Вариант строки strl, набранный прописными буквами: программировать в .net лучше всего на с#.

Вывод строки strl посимвольно.

Программировать в .NET лучше всего на С#.

strl == str2 strl != str3

Строка strl больше строки str3

Индекс первого вхождения подстроки <0дин>: О

Индекс    последнего вхождения    подстроки    <0дин>:    13

Прежде чем читать дальше,    обратите внимание    на то, что метод Compare () вызывается следующим образом.

result = string.Compare(strl, str3, StringComparison.CurrentCulture);

Как пояснялось ранее, метод Compare () объявляется как static, и поэтому он вызывается по имени, а не по экземпляру своего класса.

С помощью оператора + можно сцепить (т.е. объединить вместе) две строки. Например, в следующем фрагменте кода:

string strl = "Один";

string str2 = "Два";

string    str3 = "Три";

string    str4 = strl + str2 +    str3;

переменная str4 инициализируется строкой "ОдинДваТри".

И еще одно замечание: ключевое слово string является *псевдонимом* класса System. String, определенного в библиотеке классов для среды .NET Framework, т.е. оно устанавливает прямое соответствие с этим классом. Следовательно, поля и методы, определяемые типом string, относятся непосредственно к классу System. String, в который входят и многие другие компоненты.

**Некоторые общеупотребительные методы обращения со строками**

static int Compare(string *strA,* string *strB,* StringComparison *comparisonType)* Возвращает отрицательное значение, если строка *strA* меньше строки *strB* ; положительное значение, если строка *strA* больше строки *strB* ; и нуль, если сравниваемые строки равны. Способ сравнения определяется аргументом *comparisonType*

bool Equals(string *value,* StringComparison *comparisonType)*

Возвращает логическое значение true, если вызывающая строка имеет такое же значение, как и у аргумента *value.* Способ сравнения определяется аргументом *comparisonType*

int IndexOf(char *value)*

Осуществляет поиск в вызывающей строке первого вхождения символа, определяемого аргументом *value.* Применяется порядковый способ поиска. Возвращает индекс первого совпадения с искомым символом или -1, если он не обнаружен

int IndexOf(string *value,* StringComparison *comparisonType)*

.Осуществляет поиск в вызывающей строке первого вхождения подстроки, определяемой аргументом *value.* Возвращает индекс первого совпадения с искомой подстрокой или -1, если она не обнаружена. Способ поиска определяется аргументом *comparisonType*

int LastlndexOf(char *value)*

Осуществляет поиск в вызывающей строке последнего вхождения символа, определяемого аргументом *value.* Применяется порядковый способ поиска. Возвращает индекс последнего совпадения с искомым символом или -1, если он не обнаружен

int LastlndexOf(string *value,* StringComparison *comparisonType)*

Осуществляет поиск в вызывающей строке последнего вхождения подстроки, определяемой аргументом *value.* Возвращает индекс последнего совпадения с искомой подстрокой или -1, если она не обнаружена. Способ пойска определяется аргументом *comparisonType*

string ToLower(Culturelnfo. CurrentCulture *culture)*

Возвращает вариант вызывающей строки в нижнем регистре. Способ преобразования определяется аргументом *culture*

string ToUpper(Culturelnfo. CurrentCulture *culture)*

Возвращает вариант вызывающей строки в верхнем регистре. Способ преобразования определяется ар-гументом *culture*

1. Метод Copy

Копирует строку – аргумент (str1) в строку (str2). Количество элементов в строках не может быть различным.

string str1 = "информатика";

string str2;

str2 = string.Copy(str1); // копирование в ранее созданную строку

Console.WriteLine(str1);

Console.WriteLine(str2);

Результат работы программы:

информатика

информатика

Копирование можно использовать для инициализации при объявлении новой строки.

string str2= string.Copy(str); // инициализация копированием

2. Метод IndexOf

Определяет номер элемента строки (индекс) первого вхождения заданной подстроки в строке (может быть и один символ). Если нужное сочетание не найдено, то возвращается значение –1.

Форма 1:

public int IndexOf(char value)

В этой форме метода IndexOf() возвращается первое вхождение символа value в вызывающей строке. Если символ value в ней не найден, то возвращается значение –1. При таком поиске символа настройки культурной среды игнорируются. Следовательно, в данном случае осуществляется порядковый поиск первого вхождения символа.

Форма 2:

public int IndexOf(String value)

Форма 3:

public int IndexOf(String value, StringComparison comparisonType) Во второй форме поиск первого вхождения строки, обозначаемой параметром value, осуществляется с учетом культурной среды. А в третьей форме предоставляется возможность указать значение типа

StringComparison, обозначающее способ поиска. А если искомая строка не найдена, то во второй и третьей формах данного метода возвращается значение –1.

3. Метод LastIndexOf

Применяется для обнаружения последнего вхождения символа или строки в исходной строке. Если нужное сочетание не найдено, то возвращается значение –1.

Форма 1:

public int LastIndexOf(char value)

В этой форме метода LastIndexOf() осуществляется порядковый поиск, а в итоге возвращается последнее вхождение символа value в вызывающей строке или же значение –1, если искомый символ не найден.

Вторая и третья формы позволяют искать последнее вхождение одной строки в другой.

Форма 2:

public int LastIndexOf(String value)

Форма 3:

public int LastIndexOf(String value, StringComparison comparisonType)

Во второй форме метода поиск последнего вхождения строки, обозначаемой параметром value, осуществляется с учетом культурной среды. А в третьей форме предоставляется возможность указать значение типа StringComparison, обозначающее способ поиска. А если искомая строка не найдена, то во второй и третьей формах данного метода возвращается значение –1.

*Пример*

string str1 = "информатика";

string str2 = "и";

int k1 = str1.IndexOf(str2); // поиск первого вхождения буквы "и"

// в строку str2

int k2=str1.LastIndexOf(str2); // поиск последнего вхождения буквы

//"и" в строку str2

Console.WriteLine(k1);

Console.WriteLine(k2);

Результат работы программы:

0

8

Метод IndexOf выполняет поиск с учетом регистра символов. Если нужно выполнить поиск без учета регистра символов, то можно исходный текст и искомый фрагмент привести сначала к одному и тому же регистру с помощью функций ToLower(), ToUpper() и только потом выполнить поиск методом IndexOf.

**Задача 12.1.** Использование методов обработки строк

*Программный код:*

**using System;**

**namespace C81**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Продемонстрировать поиск в строке**

**string str="C# обладает эффективными средствами обработки строк";**

**int idx;**

**Console.WriteLine("Строка str: " + str);**

**idx=str.IndexOf('о');**

**Console.WriteLine("\nИндекс первого вхождения символа 'о':" + idx);**

**idx = str.LastIndexOf('о');**

**Console.WriteLine("\nИндекс последнего вхождения символа 'о':"+ idx);**

**idx = str.IndexOf("ми", StringComparison.Ordinal);**

**Console.WriteLine("\nИндекс первого вхождения подстроки \"ми\":"+**

**idx);**

**idx = str.LastIndexOf("ми", StringComparison.Ordinal);**

**Console.WriteLine("\nИндекс последнего вхождения подстроки**

**\"ми\": " + idx);**

**char[] chrs = {'а', 'б','в' };**

**idx = str.IndexOfAny(chrs);**

**Console.WriteLine("\nИндекс первого вхождения символов " + " 'а',**

**'б' или 'в': " + idx);**

**if(str.StartsWith("C# обладает", StringComparison.Ordinal))**

**Console.WriteLine("\nСтрока str начинается с подстроки**

**\"C# обладает\"");**

**if(str.EndsWith("строк", StringComparison.Ordinal))**

**Console.WriteLine("\nСтрока str оканчивается подстрокой \"строк\"");**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.1.



Рисунок 12.1 – Результаты решения задачи 12.1

4. Методы StartsWith() и EndsWith

public bool StartsWith(string value)

public bool EndsWith(string value)

Метод StartsWith() возвращает логическое значение true, если вызывающая строка начинается с подстроки, переданной ей в качестве аргумента value. А метод EndsWith возвращает логическое значение true, если вызывающая строка оканчивается подстрокой, переданной ему в качестве аргумента value. В противном случае оба метода возвращают логическое значение false. В обоих этих методах поиск осуществляется с учетом культурной среды. Для того чтобы указать конкретный способ поиска подстроки можно воспользоваться приведенными ниже вариантами этих методов с дополнительным параметром типа StringComparison.

public bool StartsWith(string value, StringComparison comparisonType)

public bool EndsWith(string value, StringComparison comparisonType)

В этих формах можно явно указать конкретный способ поиска.

5. Метод ToLower() – возвращает вариант вызывающей строки в нижнем регистре (делает строчными все буквы в вызывающей строке)

public string ToLower

6. Метод ToUpper() – возвращает вариант вызывающей строки в верхнем регистре (все буквы в вызывающей строке делает прописными)

public string ToUpper

*Пример*

text = text.ToUpper;

7. Метод Substring() – для получения части строки. Ниже приведены две ее формы.

public string Substring (int startIndex)

public string Substring (int startIndex, int length)

В первой форме метода Substring()подстрока извлекается, начиная с места, обозначенного параметром startIndex, и до конца вызывающей строки.

Во второй форме данного метода извлекается подстрока, состоящая из количества символов, определяемых параметром length, начиная с места, обозначаемого параметром startIndex.

В обеих формах возвращается получающаяся в итоге подстрока.

**Задача 12.2.** Использование метода Substring()

*Программный код:*

**using System;**

**namespace C74**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**string str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";**

**Console.WriteLine("строка str: " + str);**

**Console.WriteLine();**

**// выделение подстроки**

**Console.Write("подстрока str.Substring(15) ");**

**string substr = str.Substring(15);**

**Console.WriteLine(substr);**

**Console.WriteLine();**

**// выделение подстроки**

**Console.Write("подстрока str.Substring(0, 15) ");**

**substr = str.Substring(0, 15);**

**Console.WriteLine(substr);**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.2.



Рисунок 12.2 – Результаты решения задачи 12.2

8. Метод Insert() – для вставки одной строки в другую public string Insert (int startIndex, string value)

где value обозначает строку, вставляемую в вызывающую строку по индексу startIndex. Метод возвращает получившуюся в итоге строку.

**Задача 12.3.** Использование метода Insert()

*Программный код:*

**using System;**

**namespace C76**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**string str = "это тест";**

**Console.WriteLine("Исходная строка: " + str);**

**// вставить строку**

**str = str.Insert(4, "простой ");**

**Console.WriteLine("Новая строка: " + str);**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.3.



Рисунок 12.3 – Результаты решения задачи 12.3

9. Метод Remove() – для удаления части строки. Ниже приведены две формы метода.

public string Remove (int startIndex)

public string Remove (int startIndex, int Count))

В первой форме метода Remove() удаление выполняется, начиная с места, указываемого по индексу startIndex, и продолжается до конца строки.

Во второй форме данного метода из строки удаляется количество символов, определяемое параметром Count, начиная с места, указываемого по индексу startIndex. В обеих формах возвращается получающаяся в итоге строка.

**Задача 12.4.** Использование метода Remove()

*Программный код:*

**using System;**

**namespace C71**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**string str = "это простой тест";**

**Console.WriteLine("Исходная строка: " + str);**

**// удалить часть строки**

**str = str.Remove(4,5);**

**Console.WriteLine("Новая строка: " + str);**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.4.



Рисунок 12.4 – Результаты решения задачи 12.4

10. Метод Replace() – для замены части строки. Ниже приведены две формы метода.

public string Replace (char oldChar, char newChar)

public string Replace (string oldValue, string newValue)

В первой форме метода Replace() все вхождения символа oldChar в вызывающей строке заменяются символом newChar.

Во второй форме данного метода все вхождения строки oldValue в вызывающей строке заменяются строкой newValue. В обеих формах возвращается получающаяся в итоге строка.

**Задача 12.5.** Использование метода Replace()

*Программный код:*

**using System;**

**namespace C70**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**string str = "это простой тест";**

**Console.WriteLine("Исходная строка: " + str);**

**// заменить строку**

**str = str.Replace("простой", "непростой");**

**Console.WriteLine("Новая строка: " + str);**

**// заменить символы в строке**

**str = str.Replace('е', 'о');**

**Console.WriteLine("Новая строка: " + str);**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.5



Рисунок 12.5 – Результаты решения задачи 12.5

11. Метод Contains()

Возвращает логическое значение true, если вызывающая строка содержит подстроку, обозначаемую параметром value, в противном случае – логическое значение false. Поиск указываемой подстроки осуществляется порядковым способом. Этот метод особенно полезен, если требуется только выяснить, находится ли конкретная подстрока в другой строке.

Форма метода:

public bool Contains(string value)

**Задача 12.6.** Использование метода Contains()

*Программный код:*

**using System;**

**namespace C82**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**//Применение метода Contains()**

**string str = "C# сочетает эффективность с производительностью.";**

**if(str.Contains("эффективность"))**

**Console.WriteLine("Обнаружена подстрока \"эффективность\".");**

**if (str.Contains("эффе"))**

**Console.WriteLine("Обнаружена подстрока \"эффе\".");**

**if (!str.Contains("эффективный"))**

**Console.WriteLine("Подстрока \"эффективный\" не обнаружена.");**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.6.



Рисунок 12.6 – Результаты решения задачи 12.6

12. Метод Trim() удаляет начальные и конечные пробелы (обрезка строк).

Форма 1:

public string Trim()

Форма 2:

public string Trim(params char[] trimChars)

В первой форме метода Trim() из вызывающей строки удаляются начальные и конечные пробелы.

Во второй форме этого метода удаляются начальные и конечные вхождения в вызывающей строке символов из массива trimChars. В обеих формах возвращается получающаяся в итоге строка.

13. Метод PadLeft() – заполнение строки символами слева, чтобы она имела необходимую минимальную длину. Для заполнения строки слева имеются две формы метода PadLeft().

Форма 1:

public string PadLeft(int totalWidth)

Форма 2:

public string PadLeft(int totalWidth, char paddingChar)

В первой форме метода PadLeft() вводятся пробелы с левой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth. А во второй форме данного метода символы, обозначаемые параметром paddingChar, вводятся с левой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth.

В обеих формах возвращается получающаяся в итоге строка.

14. Метод PadRight() – заполнение строки символами справа, чтобы она имела необходимую минимальную длину.

Форма 1:

public string PadRight(int totalWidth)

Форма 2:

public string PadRight(int totalWidth, char paddingChar)

В первой форме метода PadRigth() вводятся пробелы с правой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth. А во второй форме данного метода символы, обозначаемые параметром paddingChar, вводятся с правой стороны вызывающей строки, чтобы ее общая длина стала равной значению параметра totalWidth.

В обеих формах возвращается получающаяся в итоге строка.

**Задача 12.7.** Использование методов PadLeft() и PadRight()

*Программный код:*

**using System;**

**namespace C84**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Пример обрезки и заполнения строк**

**string str = "тест";**

**Console.WriteLine("Исходная строка: " + str);**

**// Заполнить строку пробелами слева**

**str = str.PadLeft(10);**

**Console.WriteLine("|" + str + "|");**

**// Заполнить строку пробелами справа**

**str = str.PadRight(20);**

**Console.WriteLine("|" + str + "|");**

**// Обрезать пробелы**

**str = str.Trim();**

**Console.WriteLine("|" + str + "|");**

**// Заполнить строку символами # слева**

**str = str.PadLeft(10, '#');**

**Console.WriteLine("|" + str + "|");**

**// Заполнить строку символами # справа**

**str = str.PadRight(20, '#');**

**Console.WriteLine("|" + str + "|");**

**// Обрезать символы #**

**str = str.Trim('#');**

**Console.WriteLine("|" + str + "|");**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.7.



Рисунок 12.7 – Результаты ре шения задачи 12.7

15. Метод Split() – разделение строк. При разделении строка разбивается на составные части.

Форма 1:

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

Форма 2:

public string[] Split(params char[] separator, int count)

В первой форме метода Split() вызывающая строка разделяется на составные части. В итоге возвращается массив, содержащий подстроки, полученные из вызывающей строки. Символы, ограничивающие эти подстроки, передаются в массиве separator. Если массив separator пуст, то в качестве разделителя подстрок используется пробел. А во второй форме данного метода возвращается количество подстрок, определяемых параметром count.

16. Метод Join() – соединение строк. При соединении строка составляется из отдельных частей

Форма 1:

public static string Join(string separator, string[] value)

Форма 2:

public static string Join(string separator, string[] value, int StartIndex, int count)

В первой форме метода Join() возвращается строка, состоящая из сцепляемых подстрок, передаваемых в массиве value. Во второй форме также возвращается строка, состоящая из подстрок, передаваемых в массиве value, но они сцепляются в определенном количестве count, начиная с элемента массива value[StartIndex].

В обеих формах каждая последующая строка отделяется от предыдущей разделительной строкой, определяемой параметром separator.

**Задача 12.8.** Применение метода Join()

*Программный код:*

**using System;**

**namespace C83**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Разделить и соединить строки**

**string str = "Один на суше, другой на море.";**

**char[] seps = {' ', '.', ',' };**

**// Разделить строку на части**

**string[] parts = str.Split(seps);**

**Console.WriteLine("Результат разделения строки: ");**

**for (int i = 0; i <= (parts.Length - 1); i++)**

**Console.WriteLine(parts[i]);**

**// А теперь соединим части строк**

**string whole = string.Join("|", parts);**

**Console.WriteLine("Результат соединения строки: ");**

**Console.WriteLine(whole);**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 12.8



Рисунок 12.8 – Результаты решения задачи 12.8

17. Метод Equals() – проверка символьных строк на равенство.

Возвращает значение True или False.

Форма 1:

public bool Equals(string value)

Возвращает логическое значение True, если вызывающая строка содержит ту же последовательность символов, что и строка value.

Выполняется порядковое сравнение с учетом регистра, но без учета культурной среды.

Форма 2:

public bool Equals(string value, StringComparison comparisonType)

Возвращает логическое значение True, если вызывающая строка содержит ту же последовательность символов, что и строка value. Параметр comparisonType определяет конкретный способ сравнения строк.

Форма 3:

public static bool Equals(string a, string b, StringComparison comparisonType)

Возвращает логическое значение True, если строка a содержит ту же последовательность символов, что и строка b. Параметр comparisonType определяет конкретный способ сравнения строк.

Тип StringComparison представляет собой перечисление, в котором определяются значения, приведенные ниже. Добавление параметра типа StringComparison расширяет возможности функций Compare() и Equals(). Это дает также возможность однозначно указывать способ предполагаемого сравнения строк. В силу имеющихся отличий между сравнением строк с учетом культурной среды и порядковым сравнением очень важно быть предельно точным в этом отношении.

**Значения, определяемые в перечислении** StringComparison

1) Ordinal – сравнение строк производится с использованием порядковых значений символов в строке. При этом лексикографический порядок может нарушиться, а условные обозначения, принятые в отдельной культурной среде, игнорируются.

2) CurrentCulture – сравнение строк производится с использованием текущих настроек параметров культурной среды.

3) CurrentCultureIgnoreCase – сравнение строк производится с использованием текущих настроек параметров культурной среды, но без учета регистра.

*По умолчанию* равенство строк определяется в методе Equals() исходя из порядковых значений символов и без учета культурной среды.

Следовательно, по умолчанию обе строки сравниваются в этом методе на абсолютное, посимвольное равенство подобно тому, как это делается в операторе ==.

18. Метод Compare()

Сравнение двух строк в лексикографическом (алфавитном) порядке.

Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и т.д.

Несмотря на большую универсальность метода Compare(), для простого порядкового сравнения символьных строк проще пользоваться методом CompareOrdinal().

19. Метод CompareTo() производит сравнение строк только с учетом культурной среды.

*Пример*

string str1;

string str2;

int k = str1.CompareTo(str2);

Сравнение строк str1 и str2:

если k=0, строки равны,

если k>0, str1>str2,

если k<0, str1<str2

Равными считаются идентичные строки. При сравнении используются коды символов. Так как коды букв – символов упорядочены в алфавитном порядке, CompareTo производит лексографическое сравнение строк, что можно использовать для сортировки по алфавиту

*массива строк*.

Для того чтобы стало понятнее, почему неизменяемые строки не являются помехой, воспользуемся еще одним методом обращения со строками: Substring(). Этот метод возвращает новую строку, содержащую часть вызывающей строки. В итоге создается новый строковый объект, содержащий выбранную подстроку, тогда как исходная строка не меняется, а следовательно, соблюдается принцип постоянства строк. Ниже приведена рассматриваемая здесь форма метода Substring ():

string Substring(int *индекс\_начала,* int *длина)*

где *индекс\_начала* обозначает начальный индекс исходной строки, а *длина —* длину выбираемой подстроки.

Ниже приведена программа, в которой принцип постоянства строк демонстрируется на примере использования метода Substring().

// Применить метод Substring().

using System;

class SubStr {

static void Main() {

string orgstr = "В C# упрощается обращение со строками.";

// сформировать подстроку

string substr = orgstr.Substring(5, 20);

Console.WriteLine("orgstr: " + orgstr);

Console.WriteLine("substr: " + substr);

}

}

Вот к какому результату приводит выполнение этой программы.

orgstr: В C# упрощается обращение со строками, substr: упрощается обращение

Как видите, исходная строка из переменной orgstr не меняется, а выбранная из нее подстрока содержится в переменной substr.

И последнее замечание: несмотря на то, что постоянство строк обычно не является ни ограничением, ни помехой для программирования на С#, иногда оказывается полезно иметь возможность видоизменять строки. Для этой цели в С# имеется класс StringBuilder, который определен в пространстве имен System.Text. Этот класс позволяет создавать строковые объекты, которые можно изменять. Но, как правило, в программировании на C# исгкмьзуется тип string, а не класс StringBuilder.