**Лекция 11. Строки**

**Цель лекции:** Формирование у студентов знаний, связанных с обработкой строк.

В результате изучения данной лекции студенты будут способны:

* Продемонстрировать понимание особенностей работы с символами
* Продемонстрировать знание способов инициализации строк
* Продемонстрировать понимание особенностей работы со строковыми массивами

**Символы**

Символьный тип char предназначен для хранения символов в кодировке Unicode. Символьная константа – это символ, заключенный в апострофы

*Пример*

‘A’ - «обычный символ»

“ \n” управляющая последовательность

*Управляющей последовательностью*, или простой *escape-последовательностью*, называют определенный символ, предваряемый обратной косой чертой.

*Примеры* управляющих последовательностей в C#

\a – звуковой сигнал;

\n – перевод строки;

\t – горизонтальная табуляция;

\v – вертикальная табуляция.

Управляющие последовательности могут использоваться и в строковых константах. Если требуется вывести несколько строк, можно объединить их в одну константу, отделив одну строку от другой символами \n.

*Пример*

“Каждое выражение в программе \n имеет определенный \n тип”

Эта константа при выводе будет выглядеть так:

Каждое выражение в программе

имеет определенный

тип

Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .NET из пространства имен System. В этом классе определены статические методы, позволяющие задать вид и категорию символа, а также преобразовать символ в верхний или нижний регистр и в число.

Таблица 11.1 *–* Основные методы класса System.Char

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| GetNumericValue | Возвращает числовое значение символа, если он является цифрой, и –1 в противном случае |
| GetUnicodeCategory | Возвращает категорию Unicode- символа. Все Unicode-символы разделены на категории, например, десятичные цифры (Decimal-DigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и др. |
| IsControl | Возвращает true, если символ является управляющим |
| IsDigit | Возвращает true, если символ является десятичной цифрой |
| IsLetter | Возвращает true, если символ является буквой |
| IsLetterOrDigit | Возвращает true, если символ является буквой или цифрой |
| IsLower | Возвращает true, если символ задан в нижнем регистре |
| IsNumber | Возвращает true, если символ является числом (десятичным или шестнадцатеричным) |
| IsPunctuation | Возвращает true, если символ является знаком препинания |
| IsSeparator | Возвращает true, если символ является разделителем |
| IsUpper | Возвращает true, если символ записан в верхнем регистре |
| IsWhiteSpace | Возвращает true, если символ является пробельным (пробел, перевод строки и возврат каретки) |
| Parse | Преобразует строку в символ (строка может состоять из одного символа) |
| ToLower | Преобразует символ в нижний регистр |
| ToUpper | Преобразует символ в верхний регистр |
| MaxValue, MinValue | Возвращают символы с максимальным и минимальным кодами (эти символы не имеют видимого представления) |

**Задача 11.1.** Применение методов класса System.Char

*Программный код:*

**using System;**

**namespace C68**

**{**

**class Program**

**{**

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

**{**

**char b='B', c='\x63', d='\u0032';**

**Console.WriteLine("{0} {1} {2}", b, c, d);**

**Console.WriteLine("{0} {1} {2}", char.ToLower(b),**

**char.ToUpper(c), char.GetNumericValue(d));**

**char a;**

**do**

**{**

**Console.Write("Введите символ: ");**

**a = Convert.ToChar(Console.ReadLine());**

**Console.WriteLine("Введен символ {0}, его код - {1}", a,**

**(int)a);**

**if (char.IsLetter(a)) Console.WriteLine("Буква ");**

**if (char.IsUpper(a)) Console.WriteLine("Верхний регистр");**

**if (char.IsLower(a)) Console.WriteLine("Нижний регистр");**

**if (char.IsControl(a)) Console.WriteLine("Управляющий");**

**if (char.IsNumber(a)) Console.WriteLine("Число ");**

**if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");**

**Console.WriteLine("\n");**

**}**

**while (a != 'q');**

**Console.ReadLine();**

**}**

**}**

**}**

Результаты расчета см. рис. 11.1.



Рисунок 11.1 – Результаты решения задачи 11.1

В операторе 1 описаны три символьные переменные. В цикле do анализируется вводимый с клавиатуры символ. При вводе используется функция преобразования строки, которая должна содержать один символ, в символ типа Char. Поскольку вводится строка, ввод каждого символа следует завершать нажатием клавиши Enter. Цикл выполняется, пока пользователь не введет символ q.

Вывод символа сопровождается его кодом в десятичном виде. Для вывода кода используется явное преобразование к целому типу.

**Строки**

При создании строкового литерала в действительности формируется строковый объект. Например, в следующей строке кода:

Console.WriteLine("В C# строки являются объектами.");

текстовая строка "В C# строки являются объектами . " автоматически преобразуется в строковый объект средствами С#.

Строка – это набор символов. Символ – это знак, для которого на компьютере установлен определенный ASCII – код. Строка заключается в кавычки, например, “Информатика”.

*Длина строки* – число символов строки, включая пробелы.

Строка может быть пустой, что обозначается двумя кавычками “”. Длина пустой строки равна нулю.

В программах строки играют важную роль. В строках формируется понятная пользователю текстовая информация. Строки содержат шаблоны форматирования результатов вычислений. Выводом строк можно манипулировать и направлять его на различные устройства: на экран, на принтер, записывать в файлы. Со строками выполняют следующие операции: клонирование, объединение / вставка, усечение / удаление, извлечение подстроки, изменение регистра символов, поиск подстроки и т.д.

В C# строки, как и массивы, являются объектами, поэтому строковый тип – ссылочный. Переменная, ссылающаяся на экземпляр строки, может изменяться, присваивая ссылку на новый объект. Сам экземпляр строки *меняться не может*. Операции, манипулирующие строками, создают новые строковые объекты и возвращают ссылки на них.

Для работы со строками предназначен тип string, он является встроенным типом C#. Ему соответствует базовый класс System.String библиотеки .NET.

*Создание строк:*

1. Самый простой способ построить символьную строку — воспользоваться строковым литералом. Например, в следующей строке кода переменной ссылки на строку str присваивается ссылка на строковый литерал.

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

В данном случае переменная str инициализируется последовательностью символов "Строки в C# весьма эффективны.”.

2. Символьную строку можно ввести с клавиатуры при помощи оператора Console.ReadLine(), в отличие от ввода единичного символа, который реализует функция Read.

String str = Console.ReadLine();

3. Объект типа string можно также создать из массива типа char

*Пример*

char [] charray = { ‘t’ , ‘e', ‘s’ , ‘t’ };

string str = new string(charray);

4. Конструктор класса String может сформировать строку, повторяя заданный символ (первый параметр) заданное число раз (второй параметр).

*Пример*

string str2 = new string(‘3’, 2);

Строка «» формируется двумя подряд записанными двойными кавычками – это так называемая *пустая строка*, длина которой равна 0.

**Задача 11.2.** Создать и вывести символьную строку

*Программный код:*

**using System;**

**namespace C69**

**{**

**class Program**

**{**

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

**{**

**char[] charray = { 'Э', 'т', 'о', ' ', 'с', 'т', 'р', 'о',**

**'к', 'а', '.'};**

**string str1 = new string(charray);**

**string str2 = "Вторая строка.";**

**Console.WriteLine(str1);**

**Console.WriteLine(str2);**

**Console.ReadLine();**

**}**

**}**

**}**

Результат работы программы см. рис. 11.2:



Рисунок 11.2 – Результаты решения задачи 11.2

**Операции, которые можно совершать над строками**

* присваивание (=);
* проверка на равенство (==);
* проверка на неравенство (!=);
* обращение по индексу ([]);
* сцепление (конкатенация) строк (+).

1. Операция конкатенации (соединения)

a) с помощью знака «+».

string str= "Язык"

string str1="программирования"

string str0=str+str1+ "C#";

Console.WriteLine(str0 + "C#");

б) при помощи функции Concat()

Форма 1:

public static string Concat(string str0, string str1)

Строка str1 присоединяется путем сцепления в конце строки str0.

Форма 2:

public static string Concat(string str0, string str1, string str2)

В данной форме функция Concat() возвращает строку, состоящую из последовательного сцепления строк str0, str1, str2).

2. Операция сравнения

Строки можно сравнивать с помощью операторов **==** и **!=**. Причем имеется в виду, что равные строки идентичны, т.е. одинаково не только количество элементов, но и их последовательный набор (т.е. все содержимое). В обоих случаях выполняется *порядковое сравнение*.

Порядковое сравнение означает, что строки сравниваются на основании двоичных значений символов, из которых они состоят.

if (str==str1)…….

if (str!=str1)…….

Строки можно также сравнивать с учетом различных особенностей культурной среды, например, в лексикографическом порядке. Это так называемое сравнение *с учетом культурной среды* (функция Equals(), где непременно нужно указать способ сравнения в виде аргумента StringComparison.CurrentCulture). Для сравнения строк существует также метод Compare(). Этот метод 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 функцией Equals() или строковыми операторами.

Кроме того, строки можно сравнивать с учетом или без учета регистра.

Доступ к символу – элементу строки производится через его *индекс*.

Элементы строки нумеруются от нуля. С помощью индекса можно только прочитать символ, но нельзя присвоить символу другое значение. Новое значение можно присвоить только всей строке.

*Пример*

string str = "информатика";

char c = str[2];

Console.WriteLine(c);

Результат работы программы:

ф

Строке может быть присвоено новое значение. Новое значение может иметь другое количество элементов.

*Пример*

string str= "программирование";

int L=str.Length;

console.writeline(str + " L=" +L);

// присваивание строке нового содержимого

string str= "программирование на языке C#";

int L=str.Length;

console.writeline(str + " L=" +L);

Результат работы программы:

программирование L=16

программирование на языке C# L=27

Строка обладает свойством Length

string str = "информатика";

int L = str.Length;

Console.WriteLine(L);

Результат работы программы:

11

Класс типа string содержит ряд методов для обращения со строками.

Массивы строк

Аналогично данным любого другого типа, строки могут быть организованы в массивы. Ниже приведен соответствующий пример.

// Продемонстрировать массивы строк.

using System;

class StringArrays { static void Main() {

string[] str = { "Это", "очень", "простой", "тест." };

Console.WriteLine("Исходный массив: "); for (int i=0; i < str.Length; i++)

Console.Write(str[i] + " ");

Console.WriteLine("\n");

// Изменить строку.

str[l] = "тоже";

str[3] = "до предела тест!";

Console.WriteLine("Видоизмененный массив: "); for (int i=0; i < str.Length; i++)

Console.Write(str[i] + " ");

}

}

Вот какой результат дает выполнение приведенного выше кода.

Исходный массив:

Это очень простой тест.

Видоизмененный массив:

Это тоже простой до предела тест!

Рассмотрим более интересный пример. В приведенной ниже программе целое число выводится словами. Например, число 19 выводится словами "один девять".

// Вывести отдельные цифры целого числа словами, using System;

class ConvertDigitsToWords { static void Main() { int num; int nextdigit; int numdigits; int[] n = new int[20];

string[] digits = { "нуль", "один", "два",

"три", "четыре", "пять",

"шесть", "семь", "восемь",

"девять" };

num =1908;

Console.WriteLine("Число: " + num);

Console.Write("Число словами: ");

nextdigit = 0; numdigits = 0;

// Получить отдельные цифры и сохранить их в массиве п.

// Эти цифры сохраняются в обратном порядке, do {

nextdigit = num % 10; n[numdigits] = nextdigit; numdigits++; num = num /10;

} while(num > 0); numdigits--;

// Вывести полученные слова.

for( ; numdigits >= 0; numdigits--)

Console.Write(digits[n[numdigits]] + " ");

Console.WriteLine() ;

}

}

Выполнение этой программы приводит к следующему результату.

Число: 1908

Число словами: один девять нуль восемь

В данной программе использован массив строк digits для хранения словесных обозначений цифр от 0 до 9. По ходу выполнения программы целое число преобразуется в слова. Для этого сначала получаются отдельные цифры числа, а затем они сохраняются в обратном порядке следования в массиве п типа int. После этого выполняется циклический опрос массива п в обратном порядке. При этом каждое целое значение из массива п служит в качестве индекса, указывающего на слова, соответствующие полученным цифрам числа и выводимые как строки.

Применение строк в операторах switch

Объекты типа string могут использоваться для управления оператором switch. Это единственный нецелочисленный тип данных, который допускается применять в операторе switch. Благодаря такому применению строк в некоторых сложных ситуациях удается найти более простой выход из положения, чем может показаться на первый взгляд. Например, в приведенной ниже программе выводятся отдельные цифры, соответствующие словам "один", "два" и "три".

// Продемонстрировать управление оператором switch посредством строк.

using System;

class StringSwitch { static void Main() {

string[] strs = { "один", "два", "три", "два", "один" };

foreach(string s in strs) { switch (s)    {

case "один":

Console.Write (1);

break; case "два":

Console.Write (2); break; case "три":

Console.Write (3); break;

}

}

Console.WriteLine ();

}

}

При выполнении этой программы получается следующий результат.

12321

**Форматирование данных числовых типов**

Для форматирования данных числовых типов в C# предусмотрен ряд методов:

1. Console.WriteLine();

2. String.Format();

3. ToString().

Форматирование осуществляется с помощью двух компонентов: *спецификаторов формата* и *поставщиков формата*.

Спецификатор формата определяет, в какой именно удобочитаемой форме будут представлены данные. Например, для вывода числового значения в экспоненциальном представлении (т.е. в виде мантиссы и порядка числа) используется спецификатор формата E.

Для того, чтобы отформатировать данные, достаточно включить спецификатор формата в метод, поддерживающий форматирование.

**Способ 1:** с помощью метода Console.WriteLine()

Для форматирования выводимых данных служит следующая форма метода WriteLine()

WriteLine("форматирующая строка", arg0, arg1, …argN);)

В этой форме аргументы метода WriteLine() разделяются запятой, а не знаком +. А форматирующая строка состоит из двух следующих элементов: обычных печатаемых символов, отображаемых в исходном виде, а также команд форматирования.

Общая форма команд форматирования:

{argnum, width: frm},

где

argnum – это номер отображаемого аргумента, начиная с нуля;

width – минимальная ширина поля;

frm – спецификатор формата.

Параметры width и frm не являются обязательными. Поэтому в своей простейшей форме команда форматирования просто указывает конкретные аргументы для отображения. Например, команда {0} указывает аргумент arg0, команда {1} – аргумент arg1 и т.д.

Если в команде форматирования указывается параметр frm, то данные отображаются в указанном формате. В противном случае используется формат, выбираемый по умолчанию. Если же в команде форматирования указывается параметр width, то выводимые данные дополняются пробелами для достижения минимально необходимой ширины поля.

*Пример*

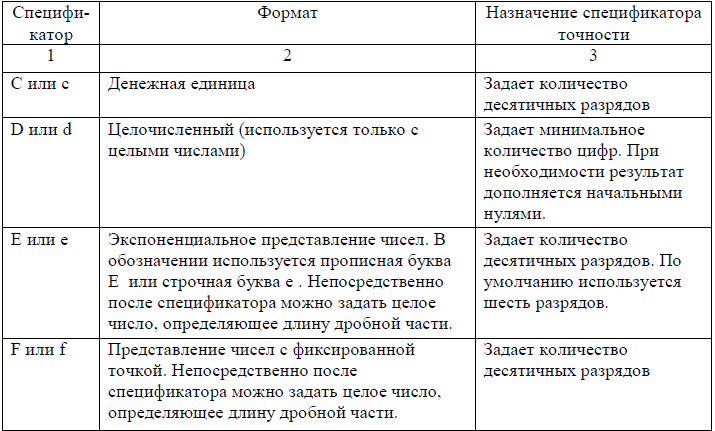
Double v=1723.56345;

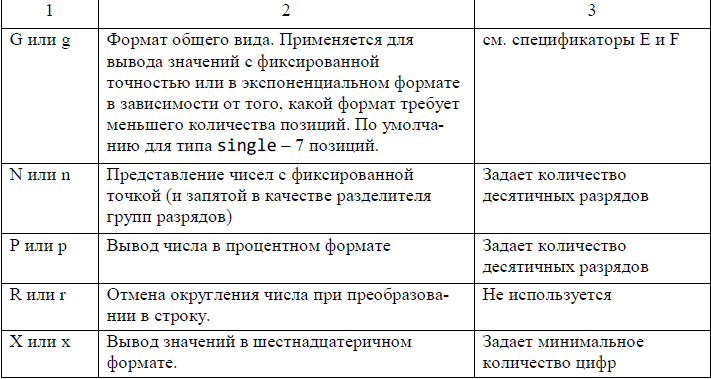
Console.WriteLine("v={0:F2}",v);

Результат:

v=1723.56

Таблица 7.2 *–* Спецификаторы формата числовых данных





**Способ 2:** применение метода String.Format() для форматирования данных.

Иногда требуется сформировать строку, содержащую отформатированные данные, но не отображать ее сразу. Это дает возможность отформатировать данные заранее, чтобы вывести их в дальнейшем на выбранное устройство. Для этого имеется два способа, один – применение метода String.Format(), а другой – в передаче спецификатора формата методу ToString().

*Пример 1*

double v=17455.34267;

string str=String.Format("{0:F2}",v);

Console.WriteLine(str);

*Пример 2*

string str=String.Format("Сумма:{0,3:D} Произведение:{1,8:D}", sum, prod);

**Способ 3:** применение метода ToString() для форматирования данных.

Здесь спецификатор формата передается методу ToString() непосредственно.

*Пример*

double v=17455.34267;

string str= v.ToString("F2");

Console.WriteLine(str);