**Лабораторная работа №ДОП-2. Работа со строками в С#.**

С точки зрения ежедневного программирования одним из самых важных типов данных С# является тип string. Он предназначен для определения и поддержки символьных строк. Во многих других языках программирования строка представляет собой массив символов. В С# дело обстоит иначе: здесь строки являются объектами. Таким образом, string – это ссылочный тип. Несмотря на то что string – встроенный тип данных, для его рассмотрения необходимо иметь представление о классах и объектах.

**Задание 1.**

**Объявление и инициализация строк. Использование простейших операций со строками.**

Класс string содержит ряд методов, которые предназначены для обработки. Тип string также включает свойство Length, которое содержит длину строки. Чтобы получить значение отдельного символа строки, достаточно использовать индекс. Например:

string str = " t e s t " ;

Console.WriteLine(string[0]);

При выполнении этого фрагмента программы на экран будет выведен символ t (первый символ слова "test"). Как и у массивов, индексация строк начинается с нуля. Однако здесь важно отметить, что с помощью индекса нельзя присвоить значение символу внутри строки (**Объяснить почему???**).

Чтобы узнать, равны ли две строки, необходимо использовать оператор "==". Обычно, когда оператор "==" применяется к ссылочным объектам, он определяет, относятся ли обе ссылки к одному и тому же объекту. Но применительно к объектам типа string дело обстоит иначе. В этом случае проверяется равенство содержимого двух строк (**Объяснить почему???**). То же справедливо и в отношении оператора " ! = " . Что касается остальных операторов отношения (например, ">" или ">="), то они сравнивают ссылки так же, как и объекты других типов.

1. Создайте консольное приложение StringExample.
2. Создайте три объекта типа string, использовав при этом три различных вида инициализации.

Например:

* string str1 = ”Строка1”;
* string str2 = new string (‘s’, 5); //объяснить синтаксис и назначение этого конструктора
* char[] charray={‘t’,’e’,’s’,’t’};

string str3 =new string (charray);

3. Протестируйте на созданных строках простейшие операции:

- присваивание (=);

- две операции проверки эквивалентности (==) и (!=);

- конкатенация или сцепление строк (+);

- взятие индекса ([]).

4. Используя индексацию строк, выполните следующие задания:

- из одной строки копировать содержание скобок комментария в новую строку (при условии того, что если скобки открываются, то они должны быть закрыты);

- скопировать в новую строку все символы другой строки за исключением слова “student”;

**Задание 2.**

**Массивы строк.**

Подобно другим типам данных строки могут быть собраны в массивы.

1. Создайте консольное приложение ArrayOfStrings;
2. Объявите функцию SumInWords, которая должна получать параметр (целое число не менее четвертого порядка), а возвращать массив строк (с наименованием валюты).

**Методические указания:**

string[] ar1 = {"один", "два",

"три", "четыре", "пять",

"шесть", "семь", "восемь",

"девять"};

И так несколько массивов для наименования валюты, десятков, сотен и т.д.

Например: Входящий параметр – 1908, результат – тысяча девятьсот восемь рублей;

Входящий параметр – 352, результат – триста пятьдесят два рубля;

**Задание 3.**

**Методы Join и Split.**

Методы Join и Split выполняют над строкой текста взаимно обратные преобразования. Динамический метод Split позволяет осуществить разбор текста на элементы. Статический метод Join выполняет обратную операцию, собирая строку из элементов.

Заданный строкой текст зачастую представляет собой совокупность структурированных элементов - абзацев, предложений, слов, скобочных выражений и т.д. При работе с таким текстом необходимо разделить его на элементы, пользуясь специальными разделителями элементов, - это могут быть пробелы, скобки, знаки препинания. Практически подобные задачи возникают постоянно при работе со структурированными текстами. Методы Split и Join облегчают решение этих задач.

Динамический метод Split, как обычно, перегружен. Наиболее часто используемая реализация имеет следующий синтаксис:

public string[] Split(params char[])

На вход методу Split передается один или несколько символов, интерпретируемых как разделители. Объект string, вызвавший метод, разделяется на подстроки, ограниченные этими разделителями. Из этих подстрок создается массив, возвращаемый в качестве результата метода. Другая реализация позволяет ограничить число элементов возвращаемого массива.

Синтаксис статического метода Join таков:

public static string Join(string delimiters, string[] items )

В качестве результата метод возвращает строку, полученную конкатенацией элементов массива items, между которыми вставляется строка разделителей delimiters. Как правило, строка delimiters состоит из одного символа, который и разделяет в результирующей строке элементы массива items; но в отдельных случаях ограничителем может быть строка из нескольких символов.

1. Создайте консольное приложение SplitJoin;

2. Написать функциию, получающую в качестве параметра строку;

3. В этой функции объявите два массива строк:

string[] SimpleSentences, Words;

SimpleSentences – должен состоять из простых предложений, составляющих основную строку;

Words – должен состоять из слов, составляющих основную строку;

4. Используя встроенный метод класса String, разделить основную строку на простые предложения, в качестве разделителя использую ‘,’ и проинициализируйте массив

SimpleSentences:

SimpleSentences = txt.Split(',');

5. Аналогично разбейте строку на слова и инициализируйте ими второй массив;

6. Используя метод Join соберите строку из элементов массива Words, проанализируйте полученный результат. (Есть ли различия между исходной строкой и результатом? Если есть, то объясните их появление)

**Задание 4.**

**Работа с объектами класса StringBuilder.**

Класс string не разрешает изменять существующие объекты. Строковый класс StringBuilder позволяет компенсировать этот недостаток. Этот класс принадлежит к изменяемым классам и его можно найти в пространстве имен System.Text. Рассмотрим класс StringBuilder подробнее.

Объекты этого класса объявляются с явным вызовом конструктора класса. Поскольку специальных констант этого типа не существует, то вызов конструктора для инициализации объекта просто необходим. Конструктор класса перегружен, и наряду с конструктором без параметров, создающим пустую строку, имеется набор конструкторов, которым можно передать две группы параметров. Первая группа позволяет задать строку или подстроку, значением которой будет инициализироваться создаваемый объект класса StringBuilder. Вторая группа параметров позволяет задать емкость объекта - объем памяти, отводимой данному экземпляру класса StringBuilder. Каждая из этих групп не является обязательной и может быть опущена. Примером может служить конструктор без параметров, который создает объект, инициализированный пустой строкой, и с некоторой емкостью, заданной по умолчанию, значение которой зависит от реализации.

Над строками этого класса определены практически те же операции с той же семантикой, что и над строками класса String:

* присваивание (=);
* две операции проверки эквивалентности (==) и (!=);
* взятие индекса ([]).

Операция конкатенации (+) не определена над строками класса StringBuilder, ее роль играет метод Append, дописывающий новую строку в хвост уже существующей.

1. Создайте консольное приложение ExampleStringBuilder;
2. Объявите функцию TestStringBuilder(), демонстрирующую главные отличия класса StringBuilder от класса String:

**Методические указания:**

- объявите два объекта класса StringBuilder, используя конструктор:

StringBuilder str1=new StringBuilder("String – example of StringBuilder");

- инициализируйте оба объекта;

- приведите примеры изменения данных объектов;

1. Вызовите эту функция через функцию Main, проанализируйте полученные результаты;

У класса StringBuilder методов значительно меньше, чем у класса String, т.к. StringBuilder дает возможность изменять строки. По этой причине у класса есть основные методы, позволяющие выполнять такие операции над строкой как вставка, удаление и замена подстрок, но нет методов, подобных поиску вхождения, которые можно выполнять над обычными строками. Технология работы обычно такова: конструируется строка класса StringBuilder; выполняются операции, требующие изменение значения; полученная строка преобразуется в строку класса String; над этой строкой выполняются операции, не требующие изменения значения строки.

**Основные методы StringBuilder:**

public StringBuilder Append (<объект>)

к строке, вызвавшей метод, присоединяется строка, полученная из объекта, который передан методу в качестве параметра;

public StringBuilder Insert (int location,<объект>)

вставляет строку, полученную из объекта, в позицию, указанную параметром location.

public StringBuilder Remove (int start, int len)

удаляет подстроку длины len, начинающуюся с позиции start;

public StringBuilder Replace (string str1,string str2)

все вхождения подстроки str1 заменяются на строку str2;

public StringBuilder AppendFormat (<строка форматов>, <объекты>)

Метод является комбинацией метода Format класса String и метода Append. Строка форматов, переданная методу, содержит только спецификации форматов. В соответствии с этими спецификациями находятся и форматируются объекты. Полученные в результате форматирования строки присоединяются в конец исходной строки.

За исключением метода Remove, все рассмотренные методы являются перегруженными. В их описании дана схема вызова метода, а не точный синтаксис перегруженных реализаций. Приведу примеры, чтобы продемонстрировать, как вызываются и как работают эти методы:

1. В функции Main сконструируйте строку, используя методы Insert и Append.
2. Затем сконструируйте строку. Выведите ее на экран пронумеровав, каждое простое предложение входящее в эту строку.

**Методические указания:**

- в цикле foreach используйте метод AppendFormat (в качестве разделителя простых предложений в составе сложного примите ‘,’):

txtbuild.AppendFormat(" {0}: {1} ", num++,sub);

1. Выведите на экран обе строки. Полученные результаты проанализируйте.

**Задание 5.**

**Емкость буфера.**

Каждый экземпляр строки класса StringBuilder имеет буфер, в котором хранится строка. Объем буфера - его емкость - может меняться в процессе работы со строкой. Объекты класса имеют две характеристики емкости - текущую и максимальную. В процессе работы текущая емкость изменяется, естественно, в пределах максимальной емкости, которая реально достаточно высока. У класса StringBuilder имеется 2 свойства и один метод, позволяющие анализировать и управлять емкостными свойствами буфера.

- свойство Capacity - возвращает или устанавливает текущую емкость буфера;

- свойство MaxCapacity - возвращает максимальную емкость буфера. Результат один и тот же для всех экземпляров класса;

- метод int EnsureCapacity (int capacity) - позволяет уменьшить емкость буфера. Метод пытается вначале установить емкость, заданную параметром capacity; если это значение меньше размера хранимой строки, то емкость устанавливается такой, чтобы гарантировать размещение строки. Это число и возвращается в качестве результата работы метода. (Объясните в каких случаях возникает потребность в уменьшении емкости буфера???)

1. Откройте консольное приложение из предыдущего задания;
2. С помощью методов Capacity и Maxcapacity измерьте текущие и максимальные емкости буфера соответственно для txtbuild и strbuild;
3. Выведите на экран полученные значения емкостей, проанализируйте полученный результат;
4. С помощью метода EnsureCapacity уменьшить емкость буфера для обоих объектов до определенного значения, и также произведите вывод на экран новых значений емкостей (объяснить разницу полученных значений);

**Задание 6.**

**Массив символов char[].**

В языке C# определен класс Char[], и его можно использовать для представления строк постоянной длины, как это делается в С++. Более того, поскольку массивы в C# динамические, то расширяется класс задач, в которых можно использовать массивы символов для представления строк. Так что имеет смысл разобраться, насколько хорошо C# поддерживает работу с таким представлением строк.

Массив char[] - это обычный массив. Его нельзя инициализировать строкой символов, как это разрешается в С++. Константа, задающая строку символов, принадлежит классу String, а в C# не определены взаимные преобразования между классами String и Char[], даже явные. У класса String есть, правда, динамический метод ToCharArray, задающий подобное преобразование. Возможно также, посимвольно передать содержимое переменной string в массив символов.

Метод ToCharArray позволяет преобразовать строку в массив символов. К сожалению, обратная операция не определена, поскольку метод ToString, которым, конечно же, обладают все объекты класса Char[], печатает информацию о классе, а не содержимое массива.

Класс Char[], как и всякий класс-массив в C#, является наследником не только класса Object, но и класса Array, и, следовательно, обладает всеми методами родительских классов. А есть ли у него специфические методы, которые позволяют выполнять операции над строками, представленными массивами символов? Таких специальных операций нет. Но некоторые перегруженные методы класса Array можно рассматривать как операции над строками. Например, метод Copy дает возможность выделять и заменять подстроку в теле строки. Методы IndexOf, LastIndexOf позволяют определить индексы первого и последнего вхождения в строку некоторого символа.

1. Создайте консольное приложение CharArray;
2. Создайте несколько строк, инициализируя их различными способами;

**Методические указания:**

char[] str1 = "Hello, World!";

string str2 = "Здравствуй, Мир!";

В каком случае выведется сообщение об ошибке и почему?

1. Напишите функцию PrintCharAr(), выводящую на экран массив символов, получаемый в качестве входящего параметра;

**Методические указания:**

В цикле произвести вывод посимвольно:

Console.Write(ar[i]);

1. Напишите функцию CharArrayToString(), обратную методу ToCharArray, т.е. преобразующую массив символов в строку;

**Методические указания:**

String CharArrayToString(char[] ar) //функция возвращает объект типа String

1. Напишите функцию IndexOfStr, находящую индекс вхождения подстроки в строку:

**Методически указания:**

int IndexOfStr( char[]s1, char[] s2) // нахождение строки s2 в s1

1. Напишите процедуру TestIndexSym(), определяющую индексы вхождения символов и подстрок в строку;
2. Проанализируйте результаты выполнения всех функций.

**Контрольные вопросы:**

1. Почему нельзя присвоить значение символу внутри строки, используя индекс?
2. Почему оператор сравнения == работает с типом String как со значимыми типами данных?
3. Перечислите методы класса String;
4. В чем заключается различие между классами String, StringBuilder и массивами символов?
5. Перечислите преимущества и недостатки класса StringBuilder.
6. Перечислите методы класса StringBuilder;
7. Объясните процесс инициализации символьного массива строкой.
8. Объясните работу и назначение свойств класса StringBuilder Capacity и MaxCapacity.