Лабораторная работа

5. Массивы и строки

Варианты заданий

Выполнить задания 1-8 по предложенным вариантам:

	№ Задания							
	1	2	3	4	5	6	7	8
Вариант 1	1.II	2.I	3.III	4.I	5.II	6.II	7.III	8.II
Вариант 2	1.I	2.III	3.II	4.III	5.I	6.I	7.I	8.I
Вариант 3	1.II	2.II	3.I	4.I	5.III	6.II	7.II	8.II
Вариант 4	1.II	2.I	3.I	4.II	5.II	6.III	7.III	8.III
Вариант 5	1.I	2.II	3.II	4.III	5.I	6.II	7.II	8.I
Вариант 6	1.III	2.III	3.I	4.II	5.II	6.I	7.I	8.II
Вариант 7	1.I	2.II	3.II	4.I	5.III	6.II	7.II	8.I
Вариант 8	1.II	2.I	3.I	4.II	5.II	6.II	7.III	8.II
Вариант 9	1.III	2.II	3.II	4.I	5.I	6.I	7.II	8.I
Вариант 10	1.I	2.III	3.I	4.II	5.II	6.III	7.I	8.III

Задание 1.

I Создать массив из 15 элементов, заполненный случайными числами в диапазоне от -50 до 50. Разработать методы:
а) расчета количества чисел, кратных трем,
б) расчета суммы положительных чисел,
в) нахождения номера максимального положительного числа,
г) определения есть ли в массиве нулевые значения,
д) вывода элементов с четными номерами по формату:
a[i]=x (i – номер элемента, x – значение).

II Создать массив из 15 элементов, заполненный случайными числами в диапазоне от -100 до 100.
Разработать методы:
а) расчета количества неотрицательных чисел,
б) расчета суммы отрицательных чисел,
в) нахождения номера минимального отрицательного числа,
г) вывода элементов с нечетными номерами по формату:
a[i]=x (i – номер элемента, x – значение).

III Создать массив из 15 элементов, заполненный случайными числами в диапазоне от -150 до 150.
Разработать методы:
а) расчета количество отрицательных чисел,
б) расчета суммы четных чисел,
в) нахождения номера максимального положительного числа,
г) вывода элементов с четными номерами по формату: a[i]=x (i – номер элемента, x – значение).

Задание 2.

I Переставьте соседние элементы массива (0-й элемент поменять с 1-м, 2-й с 3-м и т.д. Если элементов нечетное число, то последний элемент остается на своем месте).

II Циклически сдвиньте элементы массива вправо (0-й элемент становится 1-м, 1-й становится 2-м, ..., последний становится 0-м, то есть массив {3, 5, 7, 9} превращается в массив {9, 3, 5, 7}).

III Дан массив из N элементов и номер элемента в массиве k. Удалите из массива элемент с индексом k, сдвинув влево все элементы, стоящие правее элемента с индексом k.

Задание 3.

I Даны два отсортированных массива: int A[n] и int B[m]. Объедините их в один отсортированный массив int C[n+m]. Время работы алгоритма должно быть порядка n+m действий.
Например, если A={1, 4, 6, 7}, B={2, 3, 5}, то C={1, 2, 3, 4, 5, 6, 7}

II Дан массив, заполненный целыми числами от 1 до 9. Определите, сколько раз встречается в нем значение 1, 2, ..., 9. Программа должна вывести ровно 9 чисел: количество единиц, двоек, ..., девяток в данном массиве.
Например, для ввода

```
10
1 2 3 4 5 1 1 1 2 2
```

программа должна вывести

```
4 3 1 1 1 0 0 0 0
```

III Дан массив из n элементов. Посчитать, сумму положительных чисел, находящихся между минимальным и максимальным элементом.

Задание 4.

Наглядная форма записи матрицы:

```
123  10   1
 14 221  17
  1  15 125
 17   1  14
```

I Дана матрица M x N, заполненная случайными числами. Программа должна выводить матрицу в наглядном виде (см. выше). Найти минимальный элемент в каждой строке и вывести его. Используя цикл foreach, найти среднее значение для всех элементов матрицы, отбросив максимальный и минимальный.

II Дана матрица M x N, заполненная случайными числами. Программа должна выводить матрицу в наглядном виде (см. выше). Найти максимальный элемент в каждом столбце и вывести его. Используя цикл foreach, найти среднее значение среди всех нечетных элементов.

III Дана матрица M x N, заполненная случайными числами. Программа должна выводить матрицу в наглядном виде (см. выше). Найти минимальный элемент в каждом столбце и вывести его. Используя цикл foreach, найти среднее значение для всех отрицательных элементов матрицы.

Задание 5.

I Дана матрица M x N, заполненная случайными числами. Характеристика столбца представляет сумму модулей его отрицательных нечетных элементов. Переставить столбцы матрицы в соответствии с ростом их характеристик.
Программа должна выводить в наглядном виде (см. задание 4) исходную и новую матрицу, с указанием характеристик.

II Дана вещественная матрица M x N, заполненная случайными числами. Соседями элемента a_{ij} являются элементы a_{kl}:
i-1<=k<=i+1; j-1<=l<=j+1, причем (i,j)!=(k,l). Провести операцию сглаживания матрицы, состоящей в вычислении среднего арифметического соседей для каждого элемента исходной матрицы.
Программа должна выводить в наглядном виде (см. задание 4) исходную и новую матрицу.

III Дана вещественная матрица M x N, заполненная случайными числами. Соседями элемента a_{ij} являются элементы a_{kl}:
i-1<=k<=i+1; j-1<=l<=j+1, причем (i,j)!=(k,l). Подсчитать количество локальных минимумов в матрице. Локальный минимум – элемент, строго меньший всех своих соседей.

Программа должна выводить в наглядном виде (см. задание 4) исходную матрицу, и список локальных минимумов.

Задание 6. Решить для типа *string*, не используя методов класса.

I Дана строка, содержащая пробелы. Найдите, сколько в ней слов (слово – это последовательность непробельных символов, первый и последний символ строки – не пробел).

II Дана строка, содержащая пробелы. Найдите в ней самое длинное слово, выведите на экран это слово и его длину.

III Даны две строки. Определите, является ли первая строка подстрокой второй строки.

Задание 7. Решить для типа StringBuilder, не используя методов класса.

I Составить строку длины *l*, заполненную случайными символами: от "a" до "h", цифрами от 1 до 7 и знаками «?», «!» и «;». Заменить все вхождения символа «;» на «_».

II Составить строку длины *l*, заполненную случайными символами: от "j" до "r", цифрами от 0 до 5 и знаками «?», «!» и «;». Заменить все вхождения символа «!» на «_».

III Составить строку длины *l*, заполненную случайными символами: от "q" до "z", цифрами от 5 до 9 и знаками «?», «!» и «;». Заменить все вхождения символа «?» на «_».

Задание 8.

I Дан ступенчатый массив, состоящий из 3 матриц: 2x2, 3x2, 2x3, заполненных случайными числами. Определить максимум в каждом столбце и вывести сумму всех таких максимумов.

II Дан ступенчатый массив, состоящий из 3 матриц: 2x2, 3x2, 2x3, заполненных случайными числами. Определить минимум в каждой строке и вывести сумму всех таких минимумов.

III Дан ступенчатый массив, состоящий из 3 матриц: 2x2, 3x2, 2x3, заполненных случайными числами. Определить минимум в каждом столбце и вывести сумму всех таких минимумов.

Методические указания

Оглавление

5.1. Массивы в C#

Массив задает способ организации данных. Массивом называют упорядоченную совокупность элементов одного типа. Каждый элемент массива имеет индексы, определяющие порядок элементов. Число индексов характеризует размерность массива. В языке C#, как и во многих других языках, индексы задаются целочисленным типом. В других языках, например, в языке Паскаль, индексы могут принадлежать счетному конечному множеству, на котором определены функции, задающие следующий и предыдущий элемент.

К сожалению, не снято ограничение 0-базируемости, означающее, что нижняя граница массивов C# фиксирована и равна нулю. Т.е. индексирование элементов всегда начинается с нуля.

5.1.1. Объявление одномерных массивов

Объявление одномерного массива выглядит следующим образом:

```
<тип>[] <объявители>;
```

В отличие от языка C++ квадратные скобки приписаны не к имени переменной, а к типу. Они являются неотъемлемой частью определения типа, так что запись T[] следует понимать как тип, задающий одномерный массив с элементами типа T.

Как и в случае объявления простых переменных, каждый объявитель может быть именем или именем с инициализацией. В первом случае речь идет об отложенной инициализации. Нужно понимать, что при объявлении с отложенной инициализацией сам массив не формируется, а создается только ссылка на массив, имеющая неопределенное значение. Поэтому пока массив не будет реально создан и его элементы инициализированы, использовать его в вычислениях нельзя.

Варианты описания массива:

```
тип[] имя;
тип[] имя = new тип [ размерность ];
тип[] имя = { список_инициализаторов };
тип[] имя = new тип [] { список_инициализаторов };
тип[] имя = new тип [ размерность ] { список_инициализаторов };
```

Примеры описаний:

```
int[] a;                                  // 1    элементов нет
int[] b = new int[4];                     // 2    элементы равны 0
int[] c = { 61, 2, 5, -9 };               // 3    new подразумевается
int[] d = new int[] { 61, 2, 5, -9 };     // 4    размерность вычисляется,
избыточное описание
int[] e = new int[4] { 61, 2, 5, -9 };    // 5    избыточное описание
```

5.1.2. Динамические массивы

Во всех вышеприведенных примерах объявлялись статические массивы, поскольку нижняя граница равна нулю по определению, а верхняя всегда задавалась в этих примерах константой. Статические массивы скорее исключение, а на практике используются динамические массивы.

Чисто синтаксически нет существенной разницы в объявлении статических и динамических массивов. Выражение, задающее границу изменения индексов, в динамическом случае содержит переменные. Единственное требование - значения переменных должны быть определены в момент объявления. Это ограничение в C# выполняется, поскольку C# контролирует инициализацию переменных.

Рассмотрим пример, в котором описана работа с динамическим массивом:

```
static void TestDynAr()
  {
      //объявление динамического массива A1
      Console.WriteLine("Введите число элементов массива A1");
      int size = int.Parse(Console.ReadLine());
      int[] A1 = new int[size];
      Arrs.CreateOneDimAr(A1);
      Arrs.PrintAr1("A1",A1);
  }
```

Здесь верхняя граница массива определяется пользователем.

5.1.3. Многомерные массивы

Очевидно, что одномерные массивы - это частный случай многомерных. Одномерные массивы позволяют задавать такие математические структуры, как векторы, двумерные - матрицы, трехмерные – кубы данных, массивы большей размерности – многомерные кубы данных.

Размерность массива это характеристика типа. Синтаксически размерность массива задается за счет использования запятых. Вот как выглядит объявление многомерного массива в общем случае:

```
<тип>[, … ,] <объявители>;
```

Число запятых, увеличенное на единицу, и задает размерность массива. Что касается объявителей, то все, что сказано для одномерных массивов, справедливо и для многомерных. Можно лишь отметить, что хотя явная инициализация с использованием многомерных константных массивов возможна, но применяется редко из-за громоздкости такой структуры. Проще инициализацию реализовать программно, но иногда она все же применяется:

```
 int[,]matrix = {
     {1,2},
     {3,4}
};
```

Чаще всего в программах используются двумерные массивы. Варианты описания двумерного массива:

```
тип[,] имя;
тип[,] имя = new тип [ разм_1, разм_2 ];
тип[,] имя = { список_инициализаторов };
тип[,] имя = new тип [,] { список_инициализаторов };
тип[,] имя = new тип [ разм_1, разм_2 ] { список_инициализаторов };
```

Примеры описаний (один пример на каждый вариант описания):

```
int[,] a;                                   // 1  элементов нет
int[,] b = new int[2, 3];                   // 2  элементы равны 0
int[,] c = {{1, 2, 3}, {4, 5, 6}};          // 3  new подразумевается
int[,] c = new int[,] {{1, 2, 3}, {4, 5, 6}};  // 4  размерность вычисляется,
избыточное описание
int[,] d = new int[2,3] {{1, 2, 3}, {4, 5, 6}};// 5  избыточное описание
```

К элементу двумерного массива обращаются, указывая номера строки и столбца, на пересечении которых он расположен, например:

```
a[1, 4];
b[i, j];
b[j, i];
```

5.1.4. Ступенчатые массивы

Еще одним видом массивов C# являются массивы массивов, называемые также ступенчатыми массивами (jagged arrays). Такой массив массивов можно рассматривать как одномерный массив, его элементы являются массивами, элементы которых, в свою очередь снова могут быть массивами, и так может продолжаться до некоторого уровня вложенности.

Эти массивы могут применяться для представления деревьев, у которых узлы могут иметь произвольное число потомков. Таковым может быть, например, генеалогическое дерево. Вершины первого уровня - Fathers, представляющие отцов, могут задаваться одномерным массивом, так что Fathers[i] - это i-й отец. Вершины второго уровня представляются массивом массивов - Children, так что Children[i] - это массив детей i-го отца, а Children[i][j] - это j-й ребенок i-го отца. Для представления внуков понадобится третий уровень, так что GrandChildren [i][j][k] будет представлять к-го внука j-го ребенка i-го отца.

Есть некоторые особенности в объявлении и инициализации таких массивов. Если при объявлении типа многомерных массивов для указания размерности использовались запятые, то для изрезанных массивов применяется более ясная символика - совокупности пар квадратных скобок; например, int[][] задает массив, элементы которого - одномерные массивы элементов типа int.

Сложнее с созданием самих массивов и их инициализацией. Здесь нельзя вызвать конструктор new int[3][5], поскольку он не задает изрезанный массив. Фактически нужно вызывать конструктор для каждого массива на самом нижнем уровне. В этом и состоит сложность объявления таких массивов.

```
//массив массивов - формальный пример
      //объявление и инициализация
      int[][] jagger = new int[3][]
  {
        new int[] {5,7,9,11},
        new int[] {2,8},
        new int[] {6,12,4}
  };
```

Массив jagger имеет всего два уровня. Можно считать, что у него три элемента, каждый из которых является массивом. Для каждого такого массива необходимо вызвать конструктор new, чтобы создать внутренний массив. В данном примере элементы внутренних массивов получают значение, будучи явно инициализированы константными массивами.

5.1.5. Оператор foreach

Оператор foreach применяется для перебора элементов в специальным образом организованной группе данных. Массив является именно такой группой. Удобство этого вида цикла заключается в том, что нам не требуется определять количество элементов в

группе и выполнять их перебор по индексу: мы просто указываем на необходимость перебрать все элементы группы. Синтаксис оператора:

```
foreach ( тип имя in выражение ) тело_цикла
```

Имя задает локальную по отношению к циклу переменную, которая будет по очереди принимать все значения из массива выражение (в качестве выражения чаще всего применяется имя массива или другой группы данных). В простом или составном операторе, представляющем собой тело цикла, выполняются действия с переменной цикла. Тип переменной должен соответствовать типу элемента массива.

Например, пусть задан массив:

```
int[] a =  { 24, 50, 18, 3, 16, -7, 9, -1 };
```

Вывод этого массива на экран с помощью оператора foreach выглядит следующим образом:

```
foreach ( int x in a ) Console.WriteLine( x );
```

Этот оператор выполняется так: на каждом проходе цикла очередной элемент массива присваивается переменной x и с ней производятся действия, записанные в теле цикла.

5.1.6. Методы и свойства класса Array

Рассмотрим основные элементы класса.

Некоторые элементы класса Array

Элемент	Вид	Описание
Length	Свойство	Количество элементов массива (по всем размерностям)
BinarySearch	Статический метод	Двоичный поиск в отсортированном массиве
Clear	Статический метод	Присваивание элементам массива значений по умолчанию
Copy	Статический метод	Копирование заданного диапазона элементов одного массива в другой массив
GetValue	Метод	Получение значения элемента массива
IndexOf	Статический метод	Поиск первого вхождения элемента в одномерный массив
Reverse	Статический метод	Изменение порядка следования элементов на обратный
Sort	Статический метод	Упорядочивание элементов одномерного массива

5.2. Случайные числа

Для генерирования последовательного ряда случайных чисел служит класс Random.

Такие последовательности чисел оказываются полезными в самых разных ситуациях, включая имитационное моделирование. Начало последовательности случайных чисел определяется некоторым начальным числом, которое может задаваться автоматически или указываться явным образом.

В классе Random определяются два конструктора.

public Random()

public Random(int seed)

Первый конструктор создает объект типа Random, использующий системное время для определения начального числа. А во втором конструкторе используется начальное значение seed, задаваемое явным образом.

Пример генерации случайного вещественного числа в диапазоне от 0 до 1:

```
Random r=new Random(); //создаем переменную класса случайных чисел
double x=r.NextDouble();//генерируем случайное число
Console.WriteLine(x);//выводим на консоль
```

Методы класса Random

public virtual int Next() - Возвращает следующее случайное целое число, которое будет находиться в пределах от 0 до Int32 . MaxValue-1 включительно;

public virtual int Next(int maxValue) - Возвращает следующее случайное целое число, которое будет находиться в пределах от 0 до maxValue-1 включительно;

public virtual int Next(int minValue, int maxValue) - Возвращает следующее случайное целое число, которое будет находиться в пределах от minValue до maxValue-1 включительно;

public virtual void NextBytes(byte[] buffer) - Заполняет массив buffer последовательностью случайных целых чисел. Каждый байт в массиве будет находиться в пределах от 0 до Byte .MaxValue-1 включительно;

public virtual double NextDouble() - Возвращает из последовательности следующее случайное число, которое представлено в форме с плавающей точкой, больше или равно 0,0 и меньше 1,0.

Ниже приведена программа, в которой применение класса Random демонстрируется на примере создания компьютерного варианта пары игральных костей.

```
// Компьютерный вариант пары игральных костей.
using System;
class RandDice {
      static void Main() {
            Random ran = new Random();
            Console.Write(ran.Next(1, 7) + " ");
            Console.WriteLine(ran.Next(1, 7));
      }
}
```

При выполнении этой программы три раза подряд могут быть получены, например, следующие результаты.

```
5 2
4 4
1 6
```

Сначала в этой программе создается объект класса Random. А затем в ней запрашиваются два случайных значения в пределах от 1 до 6.

5.3. Символы и строки

5.3.1. Класс Char

В C# есть символьный класс char, основанный на классе System.Char и использующий двухбайтную кодировку Unicode представления символов. Для этого типа в языке определены символьные константы - символьные литералы. Константу можно задавать:

- символом, заключенным в одинарные кавычки;
- escape-последовательностью;
- Unicode-последовательностью, задающей Unicode код символа.

Вот несколько примеров объявления символьных переменных и работы с ними:

```
public void TestChar()
{
    char ch1 = 'A', ch2 = '\x5A', ch3 = '\u0058';
    char ch = new Char();
    int code; string s;
    ch = ch1;
    //преобразование символьного типа в тип int
    code = ch; ch1 = (char)(code + 1);
    //преобразование символьного типа в строку
    //s = ch;
    s = ch1.ToString() + ch2.ToString() + ch3.ToString();
    Console.WriteLine("s= {0}, ch= {1}, code = {2}",
    s, ch, code);
}
```

Три символьные переменные инициализированы константами, значения которых заданы тремя разными способами. Переменная ch объявляется в объектном стиле, используя new и вызов конструктора класса.

У класса char имеются собственные методы и свойства

Статические методы и свойства класса char	
Метод	**Описание**
GetNumericValue	Возвращает численное значение символа, если он является цифрой, и (-1) в противном случае.
GetUnicodeCategory	Все символы разделены на категории. Метод возвращает Unicode категорию символа. Ниже приведен пример.
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 (десятичной или шестнадцатеричной цифрой).
IsPunctuation	Возвращает true, если символ является знаком препинания.
IsSeparator	Возвращает true, если символ является разделителем.
IsSurrogate	Некоторые символы Unicode с кодом в интервале [0x1000, 0x10FFF] представляются двумя 16-битными "суррогатными" символами. Метод возвращает true, если символ является суррогатным.
IsUpper	Возвращает true, если символ задан в верхнем регистре.

IsWhiteSpace	Возвращает true, если символ является "белым пробелом". К белым пробелам, помимо пробела, относятся и другие символы, например, символ конца строки и символ перевода каретки.
Parse	Преобразует строку в символ. Естественно, строка должна состоять из одного символа, иначе возникнет ошибка.
ToLower	Приводит символ к нижнему регистру.
ToUpper	Приводит символ к верхнему регистру.
MaxValue, MinValue	Свойства, возвращающие символы с максимальным и минимальным кодом. Возвращаемые символы не имеют видимого образа.

Большинство статических методов перегружены. Они могут применяться как к отдельному символу, так и к строке, для которой указывается номер символа для применения метода.

В языке C# определен класс char[], и его можно использовать для представления строк постоянной длины, как это делается в C++. Более того, поскольку массивы в C# динамические, 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, элементы которого являются символами. Массив символов можно преобразовать в строку, можно выполнить и обратное преобразование. У класса string есть конструктор, которому в качестве аргументов можно передать массив символов. У класса string есть динамический метод ToCharArray, преобразующий строку в массив символов.

Класс char[], как и всякий класс-массив в C#, является наследником не только класса object, но и класса Array. Некоторые 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. К сожалению, их нельзя использовать для более интересной операции - нахождения индекса вхождения подстроки в строку. При необходимости такую процедуру можно написать самостоятельно.

5.3.2. Класс String

Основным типом при работе со строками является тип string, задающий строки переменной длины. У класса string достаточно много конструкторов. Они позволяют сконструировать строку из:

- символа, повторенного заданное число раз;
- массива символов char[];
- части массива символов.

Рассмотрим примеры объявления строк с вызовом разных конструкторов:

```
public void TestDeclStrings()
{
    //конструкторы
    string world = "Мир";
```

```
    //string s1 = new string("s1");
    //string s2 = new string();
    string sssss = new string('s', 5);
    char[] yes = "Yes".ToCharArray();
    string stryes = new string(yes);
    string strye = new string(yes, 0, 2);
    Console.WriteLine("world = {0}; sssss={1}; stryes={2};" +
    " strye= {3}", world, sssss, stryes, strye);
}
```

Объект world создан без явного вызова конструктора, а объекты sssss, stryes, strye созданы разными конструкторами класса string. Заметьте, не допускается явный вызов конструктора по умолчанию - конструктора без параметров. Нет также конструктора, которому в качестве аргумента можно передать обычную строковую константу. Соответствующие операторы в тексте закомментированы.

Над строками определены следующие операции:

- присваивание (=);
- две операции проверки эквивалентности (= =) и (!=);
- конкатенация или сцепление строк (+);
- взятие индекса ([]).

5.3.3. Класс StringBuilder

Класс string не разрешает изменять существующие объекты. Строковый класс StringBuilder (построитель строк) 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. Приведем в качестве примера синтаксис трех конструкторов.

```
public StringBuilder (string str, int cap);
```

Параметр str задает строку инициализации, cap - емкость объекта.

```
public StringBuilder (int curcap, int maxcap);
```

Параметры curcap и maxcap задают начальную и максимальную емкость объекта.

```
public StringBuilder (string str, int start, int len, int cap);
```

Параметры str, start, len задают строку инициализации, cap - емкость объекта.

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.

Со строкой этого класса можно работать как с массивом, но, в отличие от класса String, здесь уже все делается как надо: допускается не только чтение отдельного символа, но и его изменение.

Пример использования возможностей StringBuilder:

```
static void Main(string[] args)
{
    StringBuilder s = new StringBuilder("This is C#");
    s.Remove(8, 2);
    s.Append("Sparta.");
    s[14] = '!';
    Console.WriteLine(s.ToString());
}
```

Источники:

1. Биллиг В. Основы программирования на C#. Режим доступа: http://www.intuit.ru/studies/courses/2247/18/info
1. Биллиг В. Объектное программирование в классах на C# 3.0. Режим доступа: http://www.intuit.ru/studies/courses/1076/429/info
2. Павловская Т.А. Программирование на C#. Учебный курс. Режим доступа: http://ips.ifmo.ru/courses/csharp/index.html