Практическое занятие №16

Работа со строками.

Цель: приобретение навыков алгоритмизации и программирования задач, оперирующих строковыми типами данных:

- ввод и вывод строковых данных;
- обработка строковых данных;
- использование стандартных процедур и функций языка C# для обработки строковых данных.

Порядок выполнения работы

Теоретический материал

За представление строк в *C#* отвечает класс *System.String*. В коде, для объявления переменной соответствующего типа, предпочтительно использовать следующий вариант написания: *string* – с маленькой буквы. Это ключевое слово языка, используя которое можно объявлять строковые переменные, также как *int* является псевдонимом для *System.Int32,* а *bool* – для *System.Boolean*.

```
string s1 = "Hello, World!";
Console.WriteLine(s1);
```

Допустимо объявление строковых переменных через ключевое слово *var*:

```
var s2 = "Create by var";
Console.WriteLine(s2);
```

Для объединения строк используется оператор +:

```
string s3 = "Hello, ";
string s4 = s3 + "John!";
Console.WriteLine(s4);
```

При работе со *String* следует помнить, что при переопределении значения переменной создается новый экземпляр строковой переменной в памяти. Поэтому, если вам нужно собрать строку из большого количества составляющих, то использование оператора + не самый лучший вариант. В этом случае будет происходить перерасход памяти: при выполнении операции объединения с присваиванием для очень большого количества подстрок, приложение может аварийно завершиться из-за того, что сборщик мусора не будет успевать удалять неиспользуемые объекты, а новые будут продолжать появляться с большой скоростью. Для решения этой задачи используйте *StringBuilder (описание нже)*.

Создание и инициализация объекта класса *String*

Существует несколько способов создать объект класса *String* и проинициализировать его. Рассмотрим варианты, которые доступны в *C#*. Наиболее распространенный способ сделать эту операцию – это присвоить строковое значение переменной без явного вызова конструктора:

```
string s5 = "test1";
var s6 = "test2";
```

Для дословного представления строки, для того чтобы проигнорировать управляющие последовательности, используйте префикс @ перед значением. Сравните вывод следующей конструкции:

```
Console.WriteLine("first line\nSecond line");
```

С вариантом:

```
Console.WriteLine(@"first line\nSecond line");
```

Если требуется подготовить строковое значение с использованием набора переменных, то можно воспользоваться статическим методом *Format* класса *String,* либо префиксом *$*:

```
int age = 27;
Console.WriteLine(String.Format("Age: {0}", age));
Console.WriteLine("");
Console.WriteLine($"Age: {age}");
Console.WriteLine("");
```

Можно явно вызвать конструктор типа с передачей в него параметров. Самый простой вариант – это передать строку:

```
string s7 = new string("test3");
```

В качестве параметра может выступать массив *Char* элементов:

```
char[] charArray = {'H', 'e', 'l', 'l', 'o'};
string s8 = new string(charArray);
```

Ещё вариант – это указать элемент типа *char* и количество раз, которое его нужно повторить:

```
string s9 = new string('O', 10); // "OOOOOOOOOO"
```

Методы и свойства класса *String*

Объединение строк. Оператор +, методы *Concat* и *Join*

Сцеплять строки между собой можно с помощью оператора +, при этом, в результате объединения, будет создан новый объект:

```
string s10 = "Area";
string s11 = " 51";
Console.WriteLine("Concat by +: " + s10 + s11);
```

В составе *API*, который предоставляет *System.String*, есть метод *Concat*, который может выполнять ту же работу:

```
Console.WriteLine("Concat by Concat(): " +
string.Concat(s10, s11));
```

Метод *Concat* позволяет объединить до четырех строк через прямое перечисление. Если нужно таким образом объединить больше строковых переменных и значений, то используйте оператор +. Полезным свойством *Concat* является то, что он может принять на вход массив элементов типа *String* и объединить их:

```
string[] sArr1 = {"First ", "Second ", "Third "};
Console.WriteLine(string.Concat(sArr1));
```

Для объединения элементов с указанием разделителя используется метод *Join*. В предыдущем примере, элементы в массиве *sArr1* уже содержали пробел, это не всегда удобно, решим задачу объединения элементов, которые не содержат разделителей, с помощью *Join*:

```
string[] sArr2 = {"First", "Second", "Third"};
Console.WriteLine("Join elements in array by Join() with
space: " + string.Join(" ", sArr2));
```

В качестве разделителя можно использовать любую строку:

```
Console.WriteLine("Join elements in array by Join() with <->:
" + string.Join("<->", sArr2));
```

Поиск и извлечение элементов из строки. Оператор [], методы *IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny, Substring*

Для получения символа из строки с конкретной позиции можно использовать синтаксис подобный тому, что применяется при работе с массивами – через квадратные скобки []:

```
string s12 = "Hello";
Console.WriteLine("Get element by index s12[3]: " + s12[3]);
```

Для решения обратной задачи: поиск индекса первого (последнего) вхождения элемента или сроки в данной строке используются методы *IndexOf, IndexOfAny* и *LastIndexOf, LastIndexOfAny*.

В таблице ниже перечислены некоторые из предоставляемых *System.String* вариантов этих методов.

Метод	**Описание**
IndexOf(Char)	Возвращает индекс первого вхождения символа.
IndexOf(Char, Int32)	Возвращает индекс первого вхождения символа начиная с заданной позиции.
IndexOf(Char, Int32, Int32)	Возвращает индекс первого вхождения символа начиная с заданной позиции, проверяется указанное количество элементов.
IndexOf(String) *IndexOf(String, Int32)* *IndexOf(String, Int32, Int32)*	Назначение методов совпадает с перечисленными выше, но поиск выполняется для строки.
IndexOfAny(Char[]) *IndexOfAny(Char[], Int32)* *IndexOfAny(Char[], Int32, Int32)*	Назначение методов совпадает с перечисленными выше, но выполняется поиск индекса первого вхождения любого из переданных в массиве элементов.
LastIndexOf([Char | String]) *LastIndexOf[[Char | String], Int32)* *LastIndexOf[[Char | String], Int32, Int32)*	Возвращает индекс последнего вхождения символа или строки. Можно задавать индекс, с которого начинать поиск и количество проверяемых позиций. *[Char | String]* – означает *Char* или *String*
LastIndexOfAny(Char[]) *LastIndexOfAny(Char[], Int32)* *LastIndexOfAny(Char[], Int32, Int32)*	Возвращает индекс последнего вхождения любого из переданных в массиве элементов.Можно задавать индекс с которого начинать поиск и количество проверяемых позиций

```
// s1 = "Hello, World!"
// Поиск первого вхождения символа 'r'
Console.WriteLine("Index of \'r\': " + s1.IndexOf('r'));
// Поиск первого вхождения символа 'l' начиная с позиции 4
Console.WriteLine("Index of \'l\', start at 4: " + s1.IndexOf('l', 4));
// Поиск первого вхождения строки "World"
Console.WriteLine("Index of \"World\": " + s1.IndexOf("World"));
// Поиск первого вхождения символа из набора ['o', 'd', ',']
Console.WriteLine("Index of pos of any symbol in array: " +
s1.IndexOfAny(new char[] {'o', 'd', ','}));
// Поиск последнего вхождения символа 'l'
Console.WriteLine("Last index of \'l\': " + s1.LastIndexOf('l'));
// Поиск последнего вхождения строки "or"
Console.WriteLine("Last index of \"or\": " + s1.LastIndexOf("or"));
// Поиск последнего вхождения символа из набора ['o', 'd', ',']
Console.WriteLine("Last index of pos of any symbol in array: " +
s1.LastIndexOfAny(new char[] {'o', 'd', ','}));
```

Для определения того, содержит ли данная строка указанную подстроку, а также для проверки равенства начала или конца строки заданному значению используйте методы: *Contains, StartsWith* и *EndsWith*.

Метод	**Описание**
Contains(Char) *Contains(String)*	Возвращает *True* если строка содержит указанный символ или подстроки.
StartsWith(Char) *StartsWith(String)*	Возвращает *True* если строка начинается с заданного символа или подстроки.
EndsWith(Char) *EndsWith(String)*	Возвращает *True* если строка заканчивается на заданный символ или подстроку.

```
Console.WriteLine("Contains \"World\"? " + s1.Contains("World")); // True
Console.WriteLine("Starts with \"He\"? " + s1.StartsWith("He")); // True
Console.WriteLine("Ends with \"ld\"? " + s1.EndsWith("ld")); // False
```

Задачу извлечения подстроки из данной строки решает метод *SubString*:

Метод	**Описание**
Substring(Int32)	Возвращает подстроку начиная с указанной позиции и до конца исходной строки.
Substring(Int32, Int32)	Возвращает подстроку начиная с указанной позиции с заданной длины.

```
Console.WriteLine("Substring start at pos 7: " + s1.Substring(7)); // World!
Console.WriteLine("Substring start at pos 7 (4 chars): " + s1.Substring(7,
4)); // Worl
```

Сравнение срок

Для сравнения строк можно использовать оператор сравнения ==, при этом будут сравниваться значения строковых переменных, а не их ссылки, как это делается для других ссылочных типов.

```
string t1 = "John";
string t2 = "John";
string t3 = "Mary";
Console.WriteLine("t1 == t2: " + (t1 == t2)); // True
Console.WriteLine("t1 != t2: " + (t1 != t2)); // False
Console.WriteLine("t1 == t3: " + (t1 == t3)); // False
Для сравнения также можно использовать метод Equals, но это менее удобный
вариант:
Console.WriteLine("Equals method: t1.Equals(t2)" + t1.Equals(t2)); // True
Console.WriteLine("Equals method: t1.Equals(t3)" + t1.Equals(t3)); // False
```

Модификация срок

Класс *String* предоставляет довольно большое количество инструментов для изменения строк.

Вставка строки в исходную в заданную позицию осуществляется с помощью метода *Insert*:

```
Console.WriteLine("Insert: " + "26".Insert(1, "[4]")); // 2[4]6
```

Для приведения строки к заданной длине с выравниванием по левому (правому) краю с заполнением недостающих символов пробелами используются методы *PadLeft* и *PadRight*:

```
Console.WriteLine("PadLeft: ");
Console.WriteLine("some string".PadLeft(15)); // "  some string"
```

```
Console.WriteLine("some string".PadLeft(15, '*')); // "****some
string"
Console.WriteLine("PadRight: ");
Console.WriteLine("some string".PadRight(15)); // "some string "
Console.WriteLine("some string".PadRight(15, '*')); // "some
string****"
```

Метод *Remove* удаляет подстроку из исходной строки. Возможны два варианта использования:

Метод	**Описание**
Remove(Int32)	Удаляет все символы начиная с заданного и до конца строки.
Remove(Int32, Int32)	Удаляет с указанной позиции заданное число символов.

```
Console.WriteLine("Remove demo1: " + "Hello".Remove(2));
Console.WriteLine("Remove demo2: " + "Hello".Remove(2, 2));
```

Замена элементов строки производится с помощью метода *Replace*. Наиболее часто используемые варианты – это замена символа на символ и строки на подстроку:

```
Console.WriteLine("Hello, World!".Replace('!', '.')); // Hello, World.
Console.WriteLine("Hello, World!".Replace("World", "John")); // Hello, John!
```

Для преобразования строки к верхнему регистру используйте метод *ToUpper()*, к нижнему – *ToLower()*:

```
Console.WriteLine("Hello, World!".ToUpper()); // HELLO, WORLD!
Console.WriteLine("Hello, World!".ToLower()); // hello, world!
```

За удаление начальных и конечных символов отвечают методы, начинающиеся на *Trim* (см. таблицу ниже).

Метод	**Описание**
Trim()	Удаляет символы пробелы из начала и конца строки.
Trim(Char)	Удаляет экземпляры символа из начала и конца строки.
Trim(Char[])	Удаляет экземпляры символов из начала и конца строки.
TrimStart() *TrimStart(Char)* *TrimStart(Char[])*	Удаляет экземпляры символов из начала строки.
TrimEnd() *TrimEnd(Char)* *TrimEnd(Char[])*	Удаляет экземпляры символов из конца строки.

```
Console.WriteLine(" hello ".Trim()); // "hello"
Console.WriteLine("***hello---".Trim('*')); // "hello---"
Console.WriteLine("***hello---".Trim(new char[] {'*', '-'})); //
"hello"
Console.WriteLine(" hello ".TrimStart()); // "hello "
Console.WriteLine(" hello ".TrimEnd()); // " hello"
```

Методы и свойства общего назначения

Рассмотрим некоторые из полезных методов и свойств, которые не вошли в приведенные выше группы.

System.Length – возвращает длину строки:

```
Console.WriteLine("Hello".Length); // 5
```

System.Split() – разделяет заданную строку на подстроки, в качестве разделителя используется указанный через параметр символ (или группа символов):

```
foreach(var s in "1 2 3".Split(' '))
Console.WriteLine(s);
foreach(var s in "1 2 3-4-5-6".Split(new char[]{' ', '-'}))
Console.WriteLine(s);
```

System.Empty – возвращает пустую строку.

Задания для самостоятельной работы:

1. В заданном тексте удалить часть текста, заключенную в скобки (вместе со скобками).
2. Сколько раз в тексте встречается заданное слово (слова разделены пробелами).
3. В тексте вставить между словами вместо одного пробела запятую и пробел.
4. Определить, какой процент слов в тексте начинается на букву К. Слова разделены пробелами.
5. Написать программу, исключающую из символьной строки все цифры.
6. Определить, содержит ли данный текст символы, отличные от букв и пробела.
7. Преобразовать введенную строку так, чтобы из нее были удалены буквы с ASCII - кодами от 70 до 75.
8. Определить, имеются ли во введенной строке следующие подряд две "4".
9. Введенную строку букв и цифр преобразовать так, чтобы после каждой цифры следовал пробел.
10. Изменить введенную строку так, чтобы каждая из цифр увеличилась на 1 (9 заменить 0).
11. Преобразовать введенную строку так, чтобы сначала были расположены цифры, потом буквы.
12. Введена строка маленьких латинских букв. Преобразовать ее, превратив маленькие буквы в большие.
13. Зашифровать введенное слово, сместив все буквы на 1 позицию (последняя становиться первой).
14. Введенную строку цифр вывести, расположив в каждой подстроке по 5 цифр.
15. Преобразовать буквы введенной строки так, чтобы их ASCII-коды увеличились на 3.
16. Введены 3 строки. Подсчитать сумму цифр, кратных 3, в каждой из них (создать функцию, подсчитывающую сумму цифр, кратных 3, в строке).
17. Даны 2 строки. Определить сумму цифр в каждой из них (создать функцию, подсчитывающую сумму цифр)
18. Создать функцию пользователя, определяющую количество символов введенной строки, ASCII-коды которых >=70.
19. Создать процедуру, позволяющую изменить введенную строку, добавив справа заданное количество заданных символов.
20. Создать процедуру, позволяющую из заданной строки удалить пробелы.
21. Создать процедуру, позволяющую записывать введенное слово в зеркальном отображении.
22. Создать процедуру, которая позволяет введенный текст разбить на строки по k символов.
23. Создать процедуру, которая позволит во введенной строке через каждые n символов вставить k пробелов.
24. Создать процедуру, которая в заданном тексте заменяет слово A1 на слово A2 (длины слов не совпадают).
25. Создать процедуру, которая во введенном слове заменяет один символ другим.

26. Создать процедуру, которая в тексте убирает лишние пробелы между словами, оставив по одному.

27. Вводятся три строки. Зашифровать каждую из них, заменив все буквы "с" на "о"(создать процедуру, заменяющую в заданной строке один символ другим)

28. Преобразовать три введенные строки, чтобы после каждой цифры следовал символ ‘!’ (создать процедуру, вставляющую пробел после каждой цифры в строке)

29. Дано предложение. Все пробелы в нем заменить на символ "_"(создать соответствующую процедуру).

30. Дано предложение. Удалить из него буквы «р» и «с»(создать процедуру, удаляющую из строки заданный символ).

Контрольные вопросы:

1.Как можно описать в программе строковые данные?
2.Перечислите процедуры для обработки строк.
3.Перечислите функции для обработки строк.
4.Какие операции можно выполнять над строковыми данными?
5.Укажите область применения строковых данных.