# Лекция 4. Строки. Символы

**Что делать?**

1. Повторить теорию из прошлой лекции:
   1. различные типы циклов;
   2. одномерные массивы;
2. Ознакомиться с теорией по темам:
   1. строки (тип **string**);
   2. методы класса String для работы со строками;
   3. символы (тип **char**);
   4. массивы символов и массивы строк.

Дополнительные ресурсы:

<http://mycsharp.ru/post/16/2013_05_26_rabota_so_strokami_v_si-sharp_klass_string.html>

<http://metanit.com/sharp/tutorial/7.2.php>

1. Реализовать примеры из данного файла. Выполнять коммит тогда, когда это указано в задании, или чаще. Ответить на вопросы в тексте.
2. Выполнить задания на самостоятельную работу (в этой лекции они не разбиты по примерам, а приводятся общим списком в конце). Если выполняется дополнительное задание, то обязательно отдельно закоммитить код без него.
3. Сдать все через GitHub.

## Фамилия, имя, отчество

Пользователь вводит три строки: имя, отчество и фамилию. Программа должна вывести следующие комбинации в указанном регистре:

1. И. О. Фамилия
2. Фамилия И.О.
3. Имя Отчество ФАМИЛИЯ
4. фам\_и\_о

Отчество может отсутствовать: если в качестве отчества указана пустая строка, то соответствующие части комбинаций следует пропустить:

*Например*,

|  |  |
| --- | --- |
| Имя: Олег  Отчество: Алексеевич  Фамилия: Кузнецов  Результат:   1. О. А. Кузнецов 2. Кузнецов О.А. 3. Олег Алексеевич КУЗНЕЦОВ 4. куз\_о\_а | Имя: джон  Отчество:  Фамилия: смит  Результат:   1. Д. Смит 2. Смит Д. 3. Джон СМИТ 4. сми\_д |

### Указания к выполнению

В C# два базовых типа предназначены для работы с текстом:

* char – отдельный **символ**, пишется в одинарных кавычках (апострофах), например, ‘a’, ‘5’, ‘И’, ‘+’, ‘π’, а также специальные клавиши – Esc, Del, Enter, Backspace и др.
* string – **строка**, состоящая из последовательности символов.

Строка является, по сути, массивом символов. Поэтому мы всегда можем извлечь из нее отдельный символ по его номеру. Например,

string s = "слово";

char bukva = s[3]; //bukva = 'в'

При этом символ нельзя напрямую конвертировать в строку, необходимо использовать метод ToString().

string cat1 = 'к'; //ОШИБКА! нельзя преобразовать

string cat2 = 'к'.ToString(); //так можно

//и так можно:

string cat3 = "кот";

cat3 = cat3[1].ToString();//cat3 = "к"

Строки можно склеивать – эта операция называется **конкатенация** – двумя способами:

* знаком +
* методом Concat

Пример:

string slovo1 = "прекрасный";

string slovo2 = "осенний";

string slovo3 = "день";

string fraza1 = slovo1 + " " + slovo2 + " " + slovo3;

// fraza1 = "прекрасный осенний день"

string fraza2 = string.Concat(slovo2, " ", slovo3, " ", slovo1);

// fraza2 = "осенний день прекрасный"

Как для строк, так и для символов существует множество полезных встроенных методов. С несколькими из них мы уже познакомились в предыдущей лекции в примере с калькулятором:

* Trim() – обрезка пробелов в начале и конце строки;
* ToUpper() – перевод строки в верхний регистр (заглавные буквы);
* ToLower() – перевод строки в нижний регистр (строчные буквы).
* Какие еще символы, кроме пробелов, обрезает Trim? Как обрезать другие символы, например, знак =?

Длину строки, можно узнать так же, как и длину массива – через свойство Length.

string s = "абвгде";

int len = s.Length; //len = 6

Добавим к ним еще один метод:

* Substring(<с\_какого\_символа>, <сколько\_символов>) – выделение подстроки, начиная с указанного символа (или обрезка строки).

Второй аргумент (число символов) можно пропустить, тогда строка обрежется до конца.

string hW = "Hello, world!";

string h = hW.Substring(0, 5); // h = "Hello"

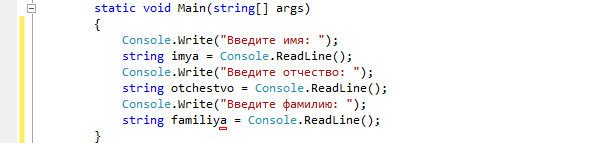
string w = hW.Substring(7, 5); // w = "world"

string w2 = hW.Substring(7); // w2 = "world!"

Этих операций достаточно, чтобы выполнить задание примера.

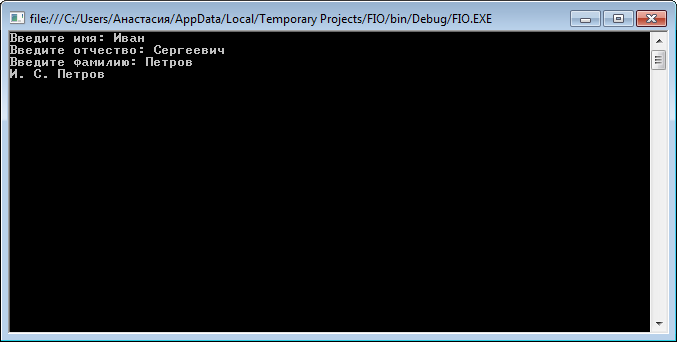
Создайте новое консольное приложение под именем FIO.

Добавьте переменные для ввода имени, отчества и фамилии:



Сформируем первую последовательность «И. О. Фамилия». Сначала сделаем это без учета регистра символов:





Как видите, это не очень сложно: нужно всего лишь извлечь первые символы из имени и отчества, и выполнить конкатенацию с фамилией и точками.

Однако, для преобразования регистра нам потребуется «разрезать» фамилию на первую букву (заглавная) и все остальные буквы (строчные). А буквы имени и отчества надо перевести в тип string.

C# позволяет использовать «многоярусные» конструкции, т.е. нам не нужно сохранять промежуточные переменные:

string i0 = imya[0].ToString();

i0 = i0.ToUpper();

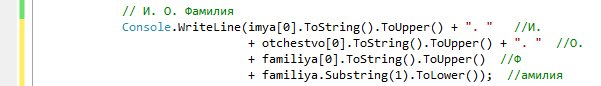
а можно сразу написать:

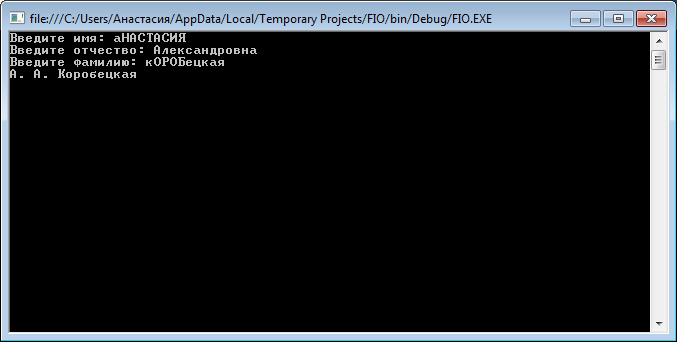
imya[0].ToString().ToUpper()

Чтобы вырезать из фамилии все буквы, кроме первой (строго говоря, нулевой), воспользуемся Substring:

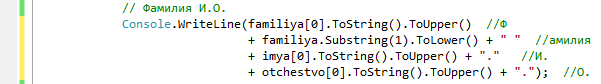
familiya.Substring(1)

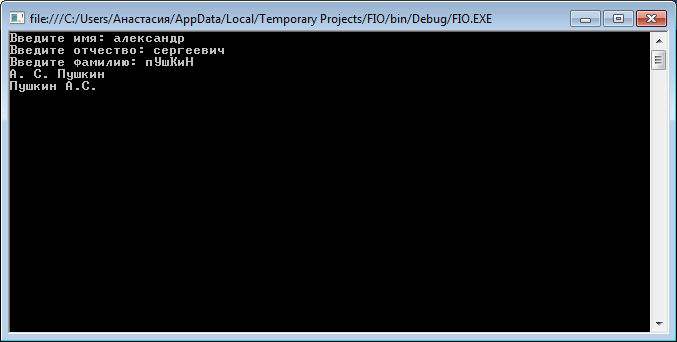
Для повышения читаемости кода, запишем каждую часть на отдельной строке с комментарием:



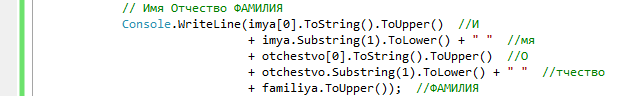


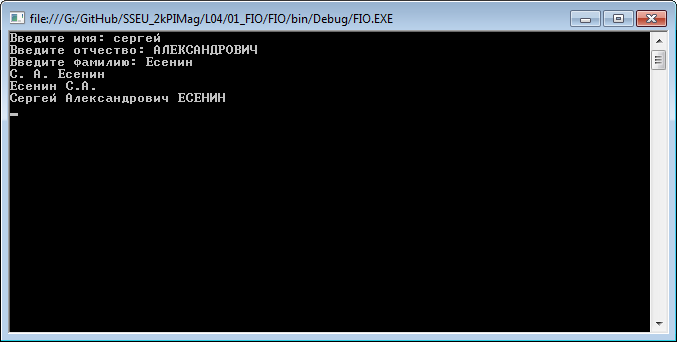
Аналогично «собирается» вторая конструкция «Фамилия И.О.», только в другом порядке:





Третья строка «Имя Отчество ФАМИЛИЯ» составляется в целом аналогично. Имя и отчество переводим так, как до этого фамилию, а фамилию – целиком в верхний регистр.





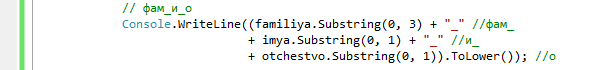
Последнюю строку «фам\_и\_о» составляем из текста только в нижнем регистре, причем из фамилии необходимо вырезать первые три символа.

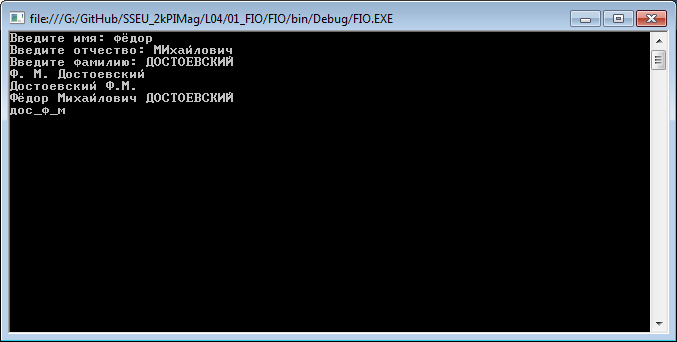
familiya.Substring(0, 3)

Поскольку весь текст нужно перевести в одинаковый (нижний) регистр, мы немного изменим порядок действий: сначала «склеим» строку, а потом к тому, что получится, применим ToLower. Для этого придется добавить скобки вокруг всей строки, например:

(imya + " " + familiya).ToLower() //имя фамилия

И еще одно отличие – чтобы избежать проблем при конкатенации символов, вместо индекса imya[0] будем использовать imya.Substring(0, 1) – эта конструкция также вырежет 1 символ, но в формате string.





* Как получить последнюю строку без использования Substring, а только через индексы символов?

Выполните коммит получившейся программы с комментарием «Л4.П1 Фамилия, имя, отчество».

Теперь добавим проверку, было ли введено отчество.

Сначала нужно разобраться, что значит «не было введено» и как это проверить.

До того, как мы присвоим переменной значение, она считается равной null, т.е. пустой, не заданной (строки вообще нет). А если пользователь в консоли нажал Enter, не введя ни одного символа, то мы получим пустую строку "", т.е. строку, в которой нет символов (но сама строка есть!).

string s; //переменная пустая (null)

s = ""; //переменная равна пустой строке

С пустой строкой можно сравнивать, как и с любой другой:

if (s != "") //если строка не пустая

Кроме того, в типе string имеется статичный метод IsNullOrEmpty, который проверяет сразу оба варианта.

if (string.IsNullOrEmpty(s)) //если строка не пустая

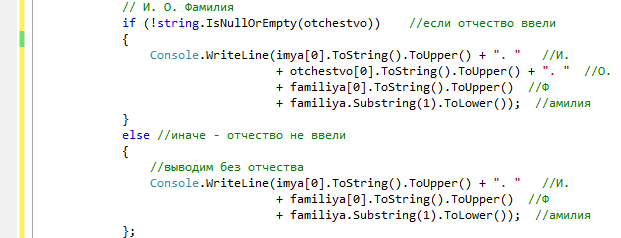
Замечание – «Статичный метод» значит, что мы вызываем его от самого типа string, а не от переменной s.

Таким образом, для отчества, которое хранится в переменной otchestvo нужно проверить:

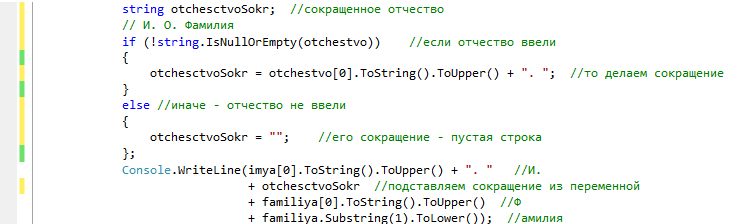
if (!string.IsNullOrEmpty(otchestvo)) //если строка не пустая

Второе решение, которое нам нужно принять – где именно выполнять эту проверку. Как всегда, есть разные варианты решения:

* выполнить проверку и вывести нужный вариант:



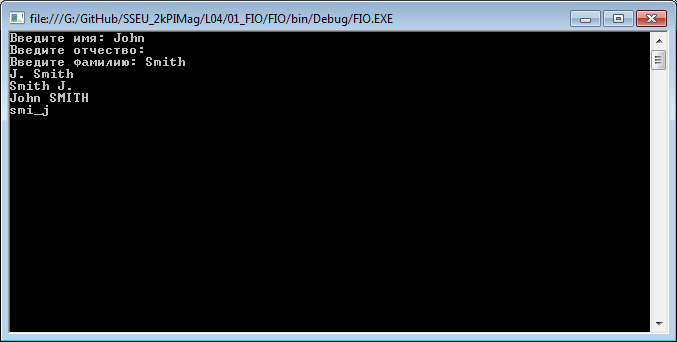
* использовать отдельную переменную, в которую будем записывать как должно выглядеть сокращенное отчество, либо пустую строку, а потом подставлять его в ответ:



Выполните проверку тем способом, который вам удобнее. Самостоятельно добавьте аналогичную проверку для всех остальных сокращений.

* Можно ли обойтись одним if?

Результат работы программы, если отчество не введено:



Выполните коммит получившейся программы с комментарием «Л4.П1 Фамилия, имя, отчество, если нет отчества».

## Идентификатор

Проверить, является ли введенная строка корректным идентификатором (именем объекта (переменной, константы и т.п.) в программе).

Идентификатор должен:

1. состоять только из букв, цифр, знаков подчеркивания;
2. начинаться с буквы или знака подчеркивания (с цифры нельзя).

Замечание: в C# можно использовать буквы и из других алфавитов, например, кириллицу, но для простоты мы будем считать, что допустимы только латинские буквы.

Примеры корректных идентификаторов: x, Value1, c00c, my\_sum, Corr.

Примеры некорректных идентификаторов: 6gh, 5chisel, x#7, NN%, я.

*Дополнительное задание*. Вывести, в чем заключается ошибка и на каком символе (по номеру в строке) она допущена.

### Указания к выполнению

Для выполнения данного задания необходимо подробнее разобраться, что же такое символ char.

С точки зрения программы, отдельный символ является даже не строкой, а числом (именно поэтому символы нельзя явно преобразовывать в строки). Это число – код символа в **таблице** **Unicode** (Юникод, UTF-8).

Поэтому символы, как и числа, можно сравнивать между собой. Чем больше код символа, тем больше символ. А поскольку в Unicode все символы записываются по алфавиту, то чем дальше символ в алфавите, тем он «больше». Заглавные буквы «меньше» строчных. Из всех алфавитов первой идет латиница, поэтому ее буквы «меньше» всех остальных. В других случаях результат сравнения не так очевиден.

Например:

‘a’ < ‘b’ < ‘c’ < ... < ‘z’

‘я’ > ‘ю’ > ... > ‘в’ > ‘б’ > ‘а’

‘0’ < ‘1’ < ... < ‘9’

‘b’ > ‘1’

‘A’ < ‘a’

‘я’ > ‘z’

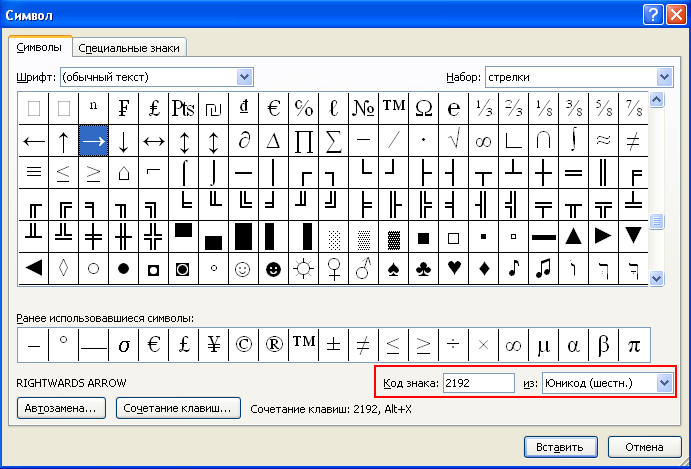
‘ч’ > ‘λ’

‘!’ < ‘0’

‘=’ < ‘≠’

‘Δ’ > ‘©’

Узнать код символа, и где он находится в таблице Unicode, можно через Word (на ленте «Вставка» → «Символ» →  «Другие символы...»). Обратите внимание, что коды символов здесь записываются в шестнадцатеричном формате. Если вы не умеете переводить шестнадцатеричные (hex) числа в обычные десятичные (dec), воспользуйтесь калькулятором Windows в режиме «Программист» или онлайн-конвертерами.



Таким образом, мы можем проверить, является ли символ, например, в переменной simvol:

* строчной буквой (от a до z):

((simvol >= 'a') && (simvol <= 'z'))

* заглавной буквой (от A до Z):

((simvol >= 'A') && (simvol <= 'Z'))

* цифрой (от 0 до 9):

((simvol >= '0') && (simvol <= '9'))

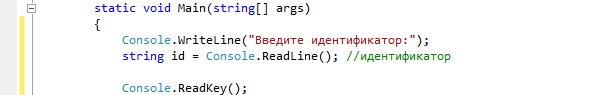
А для знака подчеркивания сравниваем на равенство:

(simvol == '\_')

Для проверки, является ли строка идентификатором, нам нужно проверить отдельно первый символ (буква или подчеркивание), а потом в цикле – все остальные символы (буква, цифра или подчеркивание). Если нам встретился недопустимый символ, то выводим сообщение об ошибке.

Создайте новое консольное приложение под именем Identificator.

Добавьте код для ввода идентификатора в переменную id:



Проверим нулевой символ на заданное условие (буква, или заглавная буква, или подчеркивание). Если условие не выполняется – нужно вывести сообщение об ошибке.

Невыполнение нескольких условий можно записать двумя способами:

* через ИЛИ (в C# ||)

ЕСЛИ (НЕ (строчная\_буква ИЛИ заглавная\_буква ИЛИ подчеркивание))

if (!((id[0] >= 'a') && (id[0] <= 'z')

|| (id[0] >= 'A') && (id[0] <= 'Z')

|| (id[0] == '\_'))

* через И (в C# &&)

ЕСЛИ (НЕ строчная\_буква И НЕ заглавная\_буква И НЕ подчеркивание)

if (!((id[0] >= 'a') && (id[0] <= 'z')

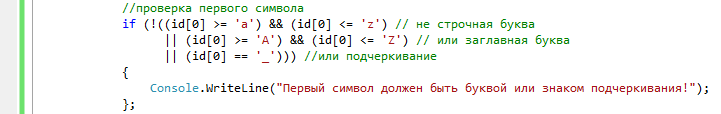
&& !((id[0] >= 'A') && (id[0] <= 'Z'))

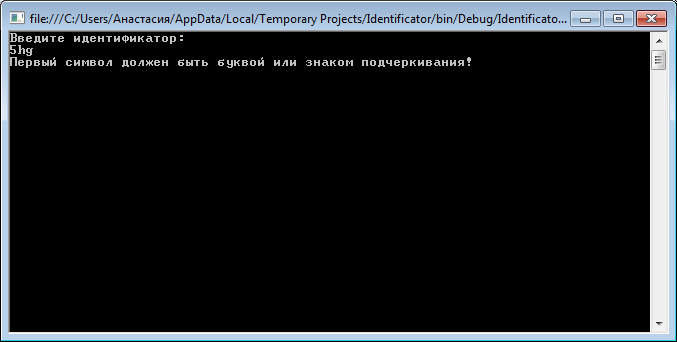
&& (id[0] != '\_'))

Второй способ длиннее, но более понятен с точки зрения русского языка.

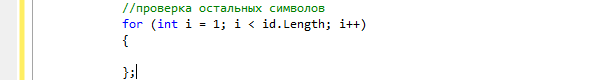
Обратите внимание на выравнивание, когда условие записано в несколько строчек: значки || и && находятся на том же уровне, что и соответствующая им открывающая скобка. Это сильно упрощает чтение сложных условий.

В примере используется первый способ, но вы можете выбрать любой:

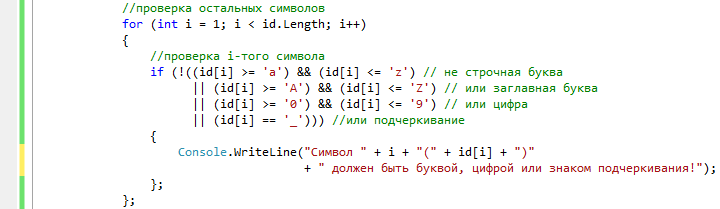


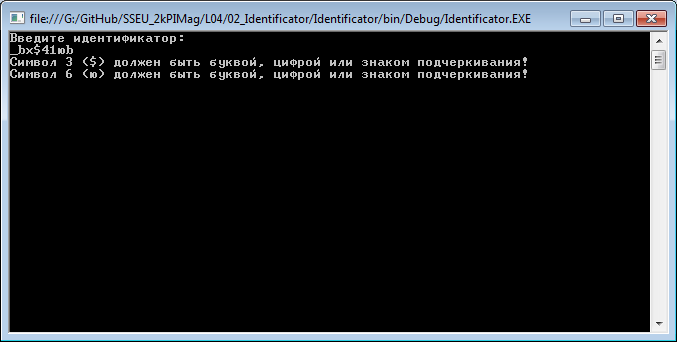


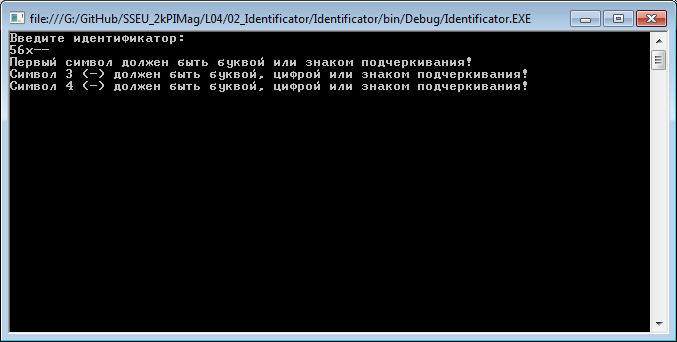
Теперь выполним проверку всех остальных символов строки, с первого (после 0) до последнего (длина строки минус 1). Воспользуемся циклом for, переменная i – номер текущего символа.



Вставьте в цикл проверку, в целом аналогичную предыдущей, только нужно добавить еще один допустимый вариант – цифры. Если условие не выполняется, то нужно указать номер символа, в котором оно нарушилось:

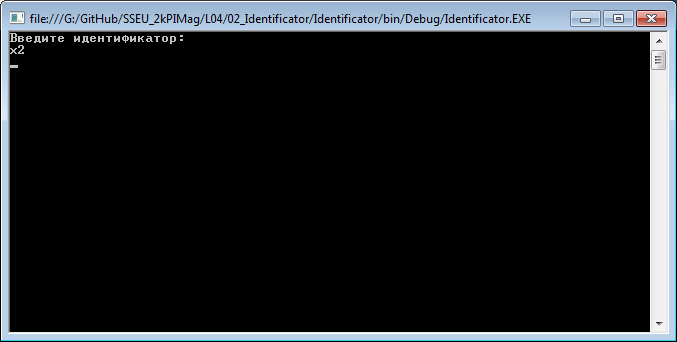






Как видите, программа сообщает об ошибках, и где именно они допущены.

Однако, если мы введем правильный идентификатор, то программа ничего не сообщит.

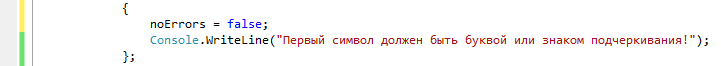


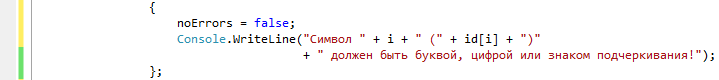
Исправим эту досадную оплошность. Нам необходимо запомнить, была ли при проверке допущена хоть одна ошибка. Если не было – то вывести сообщение, что идентификатор верный.

Лучше всего для этого подходит переменная типа bool. Объявите ее в начале программы:

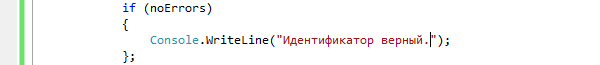


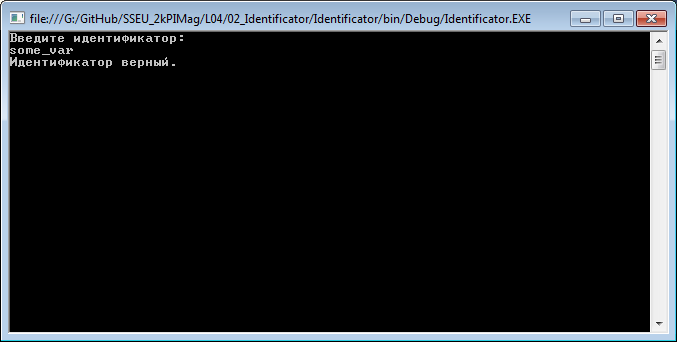
Во всех if (если возникает ошибка), сбрасываем эту переменную в false (т.е. ошибки есть).





В конце программы, если ошибок нет – выводим сообщение. Переменную типа bool не нужно ни с чем сравнивать, она сама является условием.





* Как подсчитать общее количество ошибок, если они были допущены?

Выполните коммит получившейся программы с комментарием «Л4.П2 Идентификатор».

## Извлечение чисел из строки текста

В строке содержатся числа, записанные через точку с запятой. Необходимо преобразовать их в массив чисел. Между запятыми и числами могут находиться пробелы. Если подстроку не удается преобразовать в число, то ее необходимо пропустить.

Например, входная строка: “12; 17.1; 14;2; 4и; 129.0;-5”. Выходной массив:

|  |  |
| --- | --- |
| *Индекс* | *Значение* |
|  | 12 |
|  | 17.1 |
|  | 14 |
|  | 2 |
|  | 129.0 |
|  | -5 |

### Указания к выполнению

Эту задачу выполнить проще, чем кажется на первый взгляд. В string есть встроенный метод Split (разделить, разбить) для разбивки строки на массив по заданному символу (**разделителю**, delimiter), в данном случае – по точке с запятой (;).

Таким образом, нам необходимо разбить строку, получить массив (из строк), а затем каждый элемент массива попытаться преобразовать в число.

Ранее мы обсуждали, что строка сама по себе является массивом символов. Но можно создать и массив строк, т.е. несколько строк в одной общей переменной. **Массивы строк** создаются полностью аналогично массивам чисел:

string[] colorNames = {"red", "green", "blue"}; //массив из трех строк

string[] someArray = new string[5]; //пустой массив из 5 строк

Метод Split работает следующим образом:

//строка с разделителем |

string colorList = "белый|черный|красный|синий|желтый";

//разбить на массив

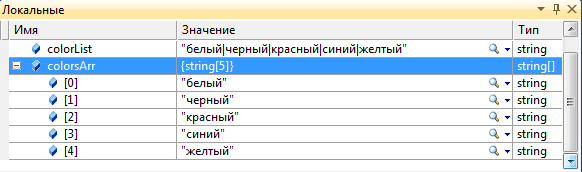
string[] colorsArr = colorList.Split('|');

Обратите внимание, разделитель '|' пишется в одинарных кавычках, т.е. является символом, а не строкой.

После выполнения этой программы массив colorsArr будет содержать:

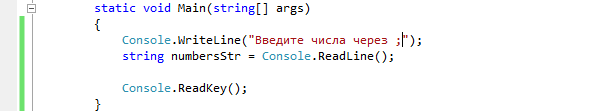
|  |  |
| --- | --- |
| *Индекс* | *Значение* |
|  | “белый” |
|  | “черный” |
|  | “красный” |
|  | “синий” |
|  | “желтый” |

Во время отладки программы в VisualStudio можно увидеть в окне Locals (Локальные):



Создайте новое консольное приложение под именем Split.

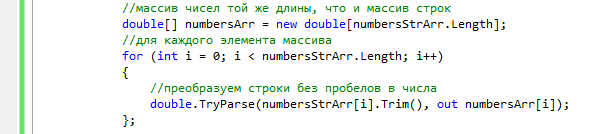
Добавьте ввод исходной строки:



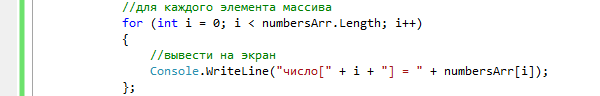
Затем преобразуем строку в массив строк:

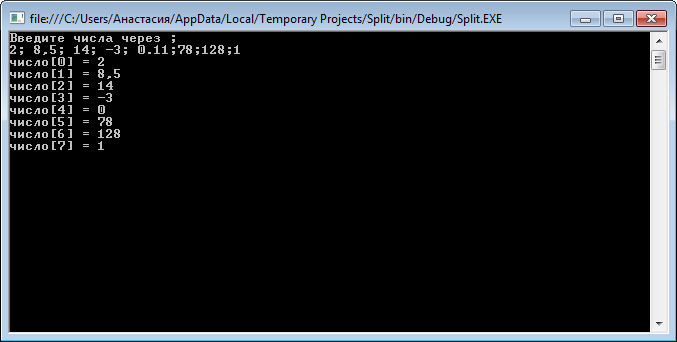


Далее необходимо массив строк преобразовать в массив чисел. В цикле отбрасываем от каждой строки в массиве пробелы и пытаемся преобразовать в число:

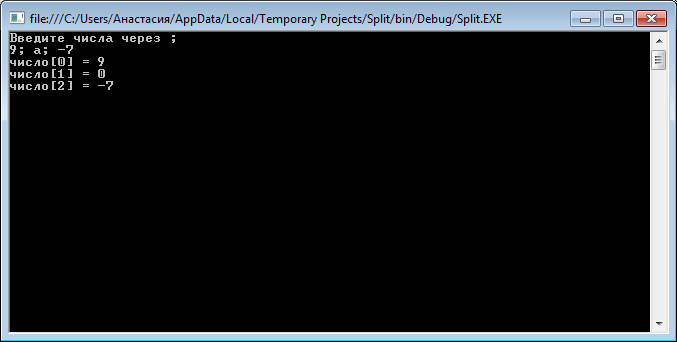


Выведем полученный массив на экран:





Задача почти решена. Однако, если мы введем неправильное число, то получим в массиве 0:



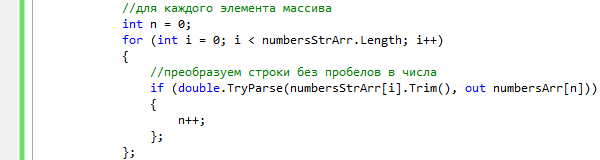
А по заданию, неправильные значения требуется пропускать. Т.е. если TryParse равно false, число не нужно добавлять в массив. При этом у нас разойдутся индекс массива строк и индекс массива чисел.

В приведенном примере:

|  |  |  |  |
| --- | --- | --- | --- |
| *Массив строк* | | *Массив чисел* | |
| *Индекс* | *Значение* | *Индекс* | *Значение* |
|  | “9” |  | 9 |
|  | “a” |  | -7 |
|  | “-7” |  |  |

В строках “-7” имеет индекс 2, а в массиве чисел -7 имеет индекс 1.

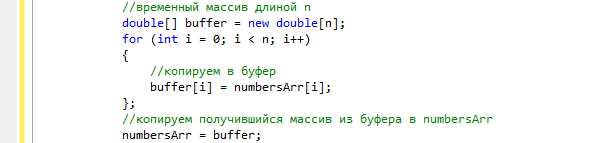
Поэтому нам потребуется дополнительная переменная (например, n), которая будет отчитывать индексы чисел. Отредактируйте цикл, преобразующий строки в числа:



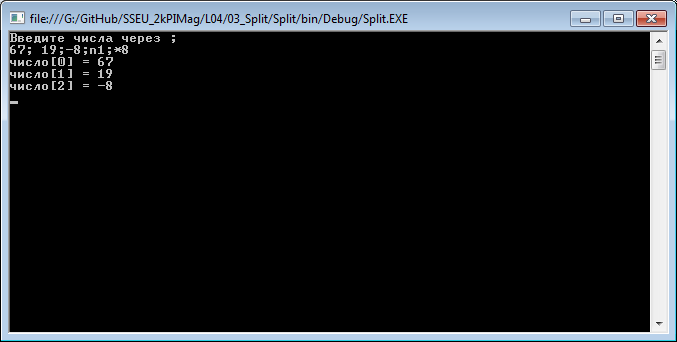
После этого в конце массива numbersArr останутся нулевые значения – столько, сколько не удалось перевести строк в числа. Эти «лишние» значения необходимо удалить.

К сожалению, массивы в C# неизменяемые, т.е. мы не можем вставить число в середину массива, или изменить его длину. Мы можем только создать новый массив, в который скопируем столько чисел, сколько нам нужно.

Создадим буферный массив (buffer), в который скопируем удачно считанные числа (n штук), а потом перенесем его обратно в numbersArr.



* Эту операцию все-таки можно выполнить одной командой. Возможно, в следующем задании, после выполнения сортировки массива, вы догадаетесь, какой.



У вас может возникнуть вопрос – а куда же делся старый numbersArr, в котором были «лишние» значения? Этот массив попадает в «мусор» (неиспользуемые данные), которые сборщик мусора (Garbage Collector), входящий в состав .NET автоматически удалит.

Выполните коммит получившейся программы с комментарием «Л4.П3 Преобразование строки в массив чисел».

## Сортировка по алфавиту

Пользователь вводит строку со словами, записанными через пробел. Необходимо отсортировать эти слова по алфавиту (без учета регистра).

Например, входная строка: “Семен абажур чемодан Чебурашка мел кольцо”.

Выходная строка: “абажур кольцо мел Семен Чебурашка чемодан”.

### Указания к выполнению

Сортировка всегда предполагает сравнение объектов в списке, будь то строки, числа, даты или какие-то другие объекты.

Мы уже рассматривали, как сравниваются между собой отдельные символы.

При сравнении строк используется их алфавитный порядок, причем строчные буквы считаются «меньше» заглавных (отличие от char). Т.е.:

1. если текст в двух строках совпадает, то они равны;

“кот” == “кот”

1. если первая строка по алфавиту идет раньше второй, то первая строка меньше:

“кит” < “кот”

“кот” < “Кот”

(хотя ‘к’ > ‘К’)

1. если первая строка по алфавиту идет позже второй, то первая строка больше:

“кот” > “кaтер”

“коТ” > “кот”

Но использовать символы < и > для сравнения строк в C# нельзя (== и != можно).

Для этого используется статический метод Compare (сравнить) класса String. Метод Compare принимает в качестве аргументов две строки и возвращает число. Если первая строка по алфавиту стоит позже второй, то возвращается число меньше нуля (–1). В противном случае возвращается число больше нуля (1). И третий случай – если строки равны, то возвращается 0.

string.Compare("кот", "кит") //вернет -1

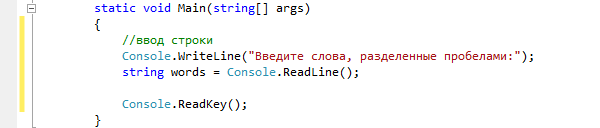
string.Compare("а", "б") //вернет 1

string.Compare("ссс", "ссс") //вернет 0

Кроме того, сортировка выполняется с массивами. Поэтому для сортировки слов по алфавиту, их надо сначала преобразовать в массив, а затем массив опять собрать в строку.

Создайте новое консольное приложение под именем SortAZ.

Добавьте в программу ввод исходной строки words.



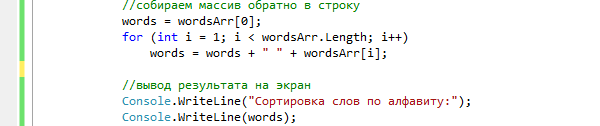
Преобразуем строку в массив с разбивкой по символу «пробел»:



На предыдущей лекции мы выполняли сортировку вручную. Теперь воспользуемся статическим методом Sort класса Array. В него уже встроен алгоритм сортировки и сравнение строк, когда сортировка выполняется по строкам, а не по словам.



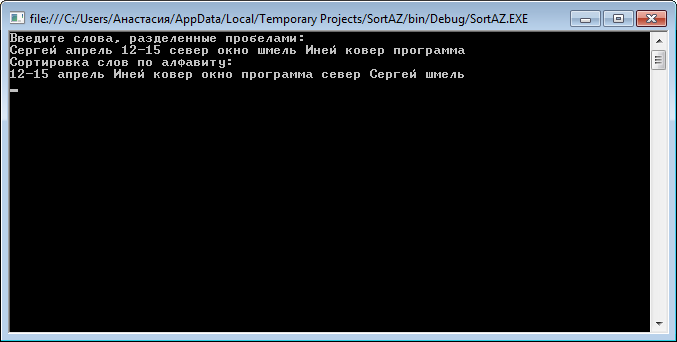
Осталось получить из массива общую строку, в которой они опять записываются через пробел, и вывести ее на экран.



Обратите внимание, в цикле for не используются фигурные скобки {} для обрамления тела цикла. Это допустимо, если в цикле выполняется всего одна команда. То же самое касается и оператора if, и цикла while.

* Можно ли написать words += " " + wordsArr[i]?

Проверка работы программы:



Выполните коммит получившейся программы с комментарием «Л4.П4 Сортировка слов по алфавиту».

## Обработка текста

Разработать программу, которая позволяет ввести текст и выполнять с ней следующие действия:

1. ввести новый текст взамен старого;
2. выполнить поиск введенной строки в тексте;
3. выполнить замену введенной строки в тексте на другую строку;
4. удалить из текста все вхождения введенной строки;
5. вставить строку в текст, начиная с указанного номера символа;
6. обрезать строку (удалить указанное число символов в начале и в конце строки).

Программа должна иметь консольный пользовательский интерфейс. Вверху должен отображаться текущий текст и число символов в нем.

Затем идет перечень доступных действий, и какими клавишами они активируются (цифрами 1-6 соответственно, Esc – выход).

Если от пользователя требуется ввести дополнительные данные, необходимо указать, что именно нужно ввести. Например, «Введите строку для поиска:».

Если пользователь нажимает Esc, программа закрывается.

### Указания к выполнению

Все действия со строками, описанные в задании, выполняются через встроенные методы класса String.

#### Поиск в строке

С помощью метода IndexOf мы можем определить индекс первого вхождения отдельного символа или подстроки в строке:

string s1 = "hello world";

char ch = 'o';

int indexOfChar = s1.IndexOf(ch); // равно 4

Console.WriteLine(indexOfChar);

string subString = "wor";

int indexOfSubstring = s1.IndexOf(subString); // равно 6

Console.WriteLine(indexOfSubstring);

Подобным образом действует метод LastIndexOf, только находит индекс последнего вхождения символа или подстроки в строку.

#### Замена текста в строке

Метод Replace принимает два аргумента – подстрока, которую нужно заменить и новая подстрока, на которую будет заменена первая:

string s = "Hello, World, Hello";

s.Replace("Hello", "Bye"); //получим "Bye, World, Bye"

Если заменить подстроку на пустую строку, то мы, фактически, удалим ее:

string s = "Hello, World, Hello";

s.Replace("o", ""); //получим "Hell, Wrld, Hell"

#### Вставка текста в строку

Метод Insert используется для вставки подстроки в строку, начиная с указанной позиции. Данный метод принимает два аргумента – позиция и подстрока. Пример:

string s = "Hello World";

s.Insert(5,","); // получим "Hello, World"

#### Обрезка строки

Метод Remove принимает один аргумент – позиция, начиная с которой обрезается строка:

string s = "Hello, World";

s.Remove(5); // получим "Hello"

В метод Remove можно передать и второй аргумент – количество обрезаемых символов. Remove(3, 5) – удалит из строки пять символов начиная с 3-го.

string s = "Hello, World";

s.Remove(3, 5); // получим "HeWorld"

#### Консольный интерфейс

Отдельная задача – это разработка пользовательского интерфейса.

При запуске программы пользователь должен увидеть приблизительно такое меню:

╔══════════════════════════╗

║ МЕНЮ ║

║ 1 - ввести новую строку ║

║ 2 - поиск ║

║ 3 - замена ║

║ 4 - удаление ║

║ 5 - вставка ║

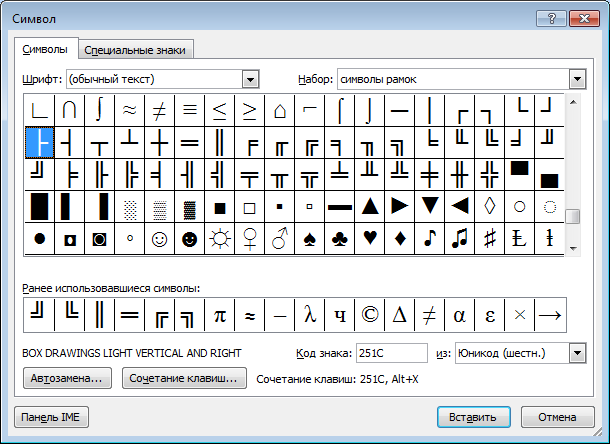
║ 6 - обрезка ║

║ ESC - выход ║

╚══════════════════════════╝

Двойная рамочка вокруг него – это не рисунок. Это особые символы, так называемая псевдографика (╔ ╝ ╚ ╗ ║ ═), применяющаяся в консольном интерфейсе.

Вы можете найти эти и другие символы в таблице Unicode:



Эти символы можно набрать в Word и скопировать в текст программы, либо задать в виде их кодов. Например, символ ╔ имеет код 2554, в C# его можно записать как:

"\u2554"

Т.е. после специальной **управляющей последовательности** \u идет код символа (в шестнадцатеричной системе, как они и отображаются в Word).

Другие управляющие последовательности:

\n – новая строка (Enter)

\t – табуляция

\' – одинарная кавычка

\" – двойная кавычка

\\ – одна обратная косая черта

\b – возврат на 1 символ назад

Кроме отображения меню, необходимо проверить, какую клавишу нажал пользователь.

Для проверки нажатой клавиши мы воспользуемся методом Console.ReadKey(), который мы раньше использовали только для завершения работы программы.

ReadKey() возвращает введенный пользователем символ или нажатую клавишу. Точнее, он возвращает тип ConsoleKeyInfo, из которого можно получить символ через свойство KeyChar

Console.ReadKey().KeyChar

Нажатие служебных клавиш Esc, Delete, Enter, F1, F2, ... F12 можно отследить по кодам, которые легко найти в Интернете. Например, код клавиши Enter – 13, Esc – 27, пробел – 32. Надо только преобразовать код (число) в символ (тип char). Можно напрямую перевести число в char:

(char)27 //ESC

При этом число записывается в обычной десятичной системе. Через управляющую последовательность это бы выглядело так (2710 = 1B16):

'\u001B'

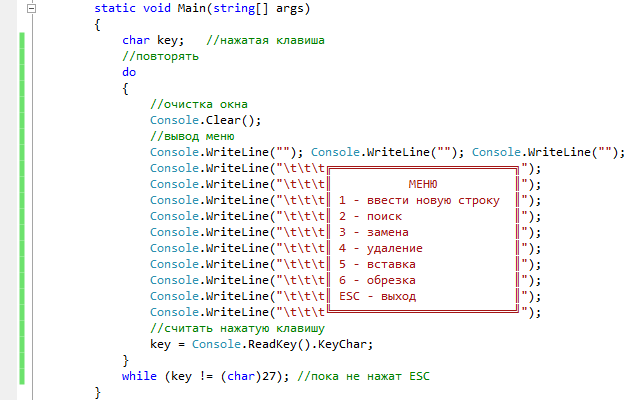
Еще одна команда, которая нам потребуется – очистка консоли от текста:

Console.Clear();

Таким образом, мы будем в цикле выводить меню и ожидать, какую клавишу нажмет пользователь. Если от 1 до 6, то будем обрабатывать строку, а если ESC – то выход. Остальные клавиши не должны делать ничего.

Сначала сформируем вывод меню и нажатие клавиши ESC для выхода.

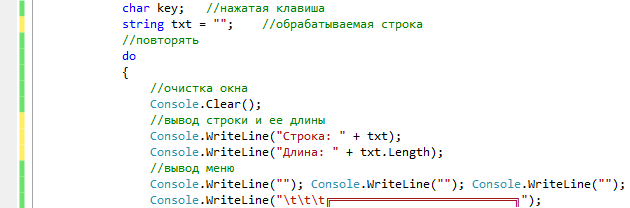
Создайте новый проект консольного приложения под именем Strings. Добавьте цикл do-while с выводом меню и проверкой нажатой клавиши.

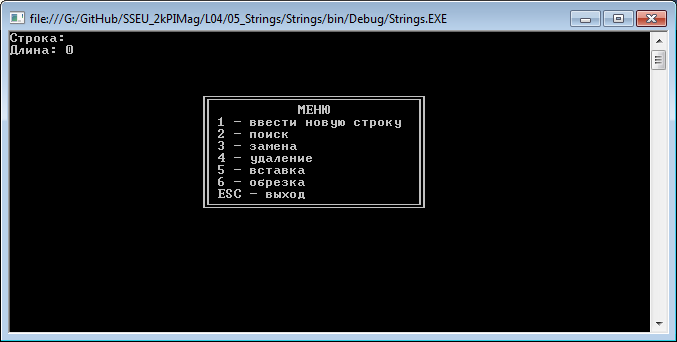


* Почему используется цикл с постусловием?

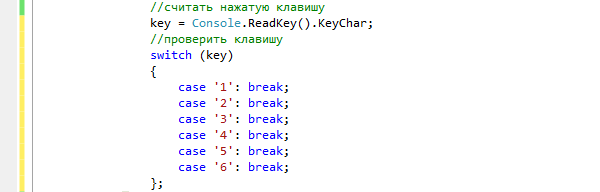
Проверьте работу программы.

Добавим переменную txt для ввода обрабатываемой строки. Пока что она будет пустой, но это не помешает вывести ее на экран и подсчитать число символов.



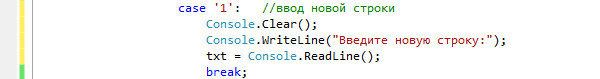


Добавим **проверку нажатых клавиш** 1-6. Обработку логичнее всего выполнять через switch, разумеется, после считывания клавиши.



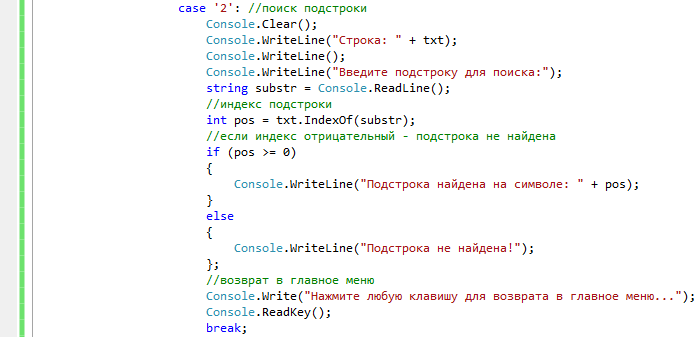
* Почему в switch отсутствует инструкция default?

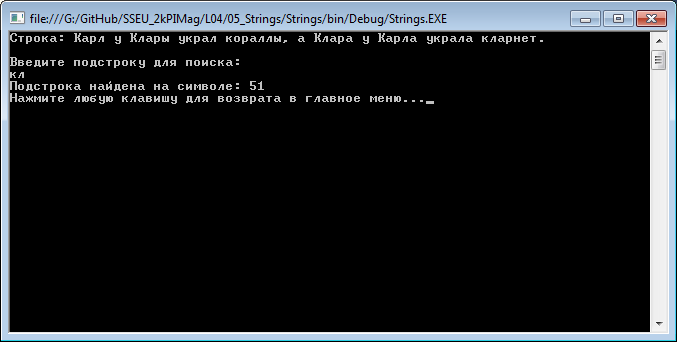
Добавим **ввод строки при нажатии кнопки 1**. При этом надо скрыть меню, чтобы показать пользователю, что в данный момент оно неактивно, и вывести приглашению к вводу:



Проверьте работу программы.

Добавим действие **по клавише 2 (поиск в строке).** Пользователь вводит подстроку, программа отображает результат и ждет команды для возвращения в главное меню.



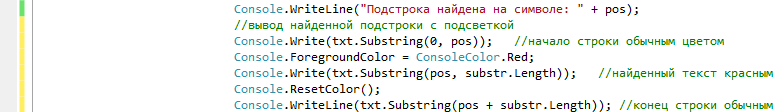


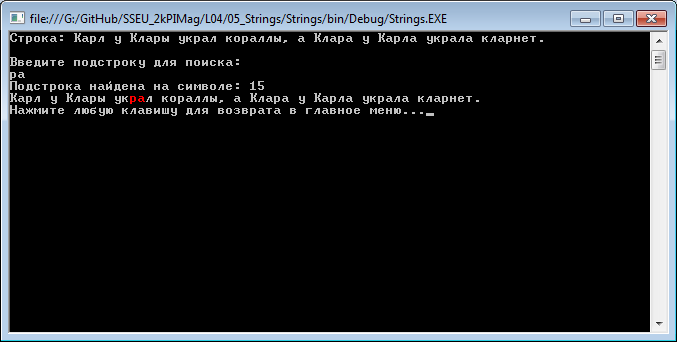
* Учитывается ли регистр символов при поиске?

*Дополнительное задание.*

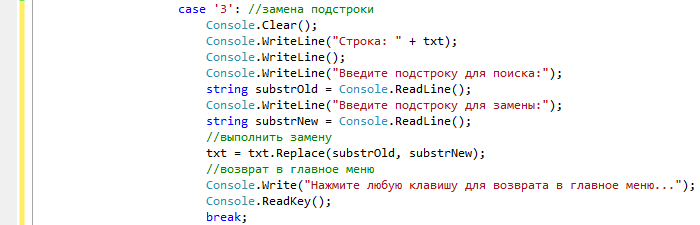
Для наглядности можно подсветить найденную подстроку другим цветом, например, красным. За цвет текста в консоли отвечает свойство ForegroundColor. Метод ResetColor позволяет вернуть цвет по умолчанию.

Таким образом, необходимо вывести начало строки до найденной позиции pos. Затем поменять цвет на красный (Red) и вывести найденную подстроку. Затем опять вернуть обычный цвет и вывести остаток строки.



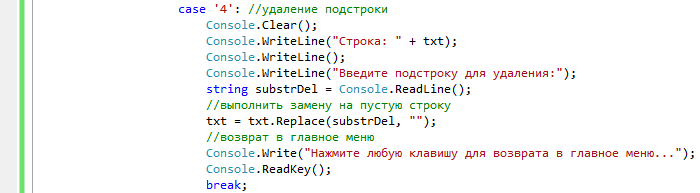


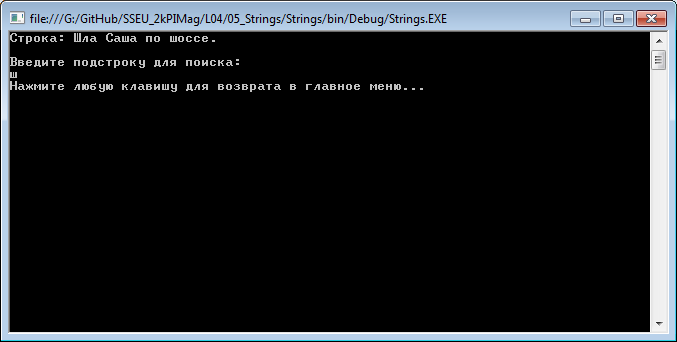
По аналогии выполним **замену (кнопка 3) подстроки**, используя метод Replace. Ввести требуется две подстроки: substrOld – какую часть строки надо заменить и substrNew – на какую новую подстроку. Обратите внимание, результат, возвращаемый методом Replace, необходимо присвоить обратно в txt, иначе он никуда сохранится.

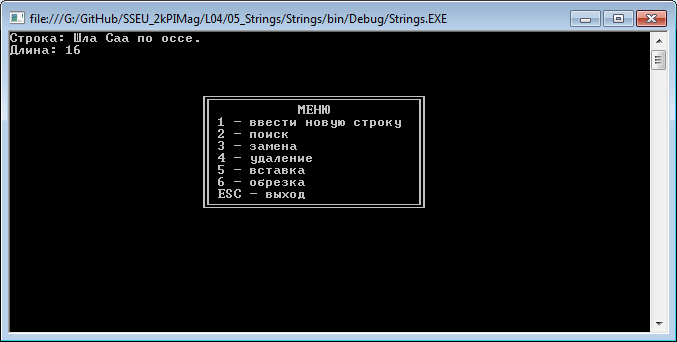


Обновленную строку мы увидим уже в главном меню. При желании, можно добавить вывод изменившейся строки и сюда. Но если вы захотите сделать подсветку, потребуется опять задействовать метод IndexOf, т.к. Replace не сообщает, где именно сделана замена.

**Удаление подстроки (кнопка 4)** очень похоже, но нам потребуется только одна подстрока для удаления.

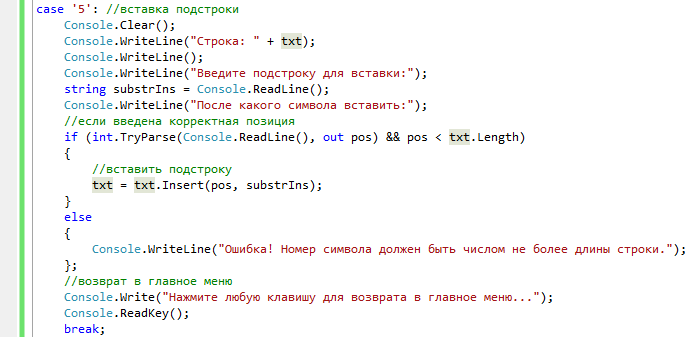




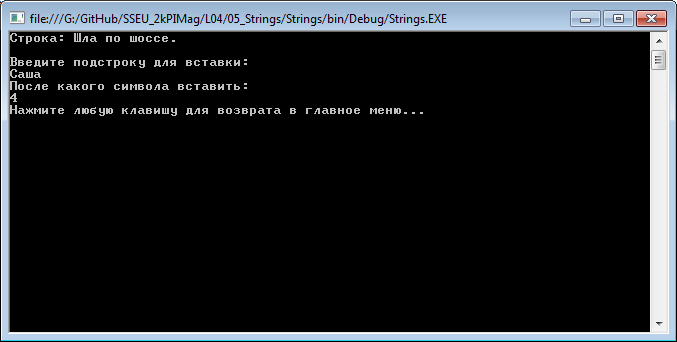


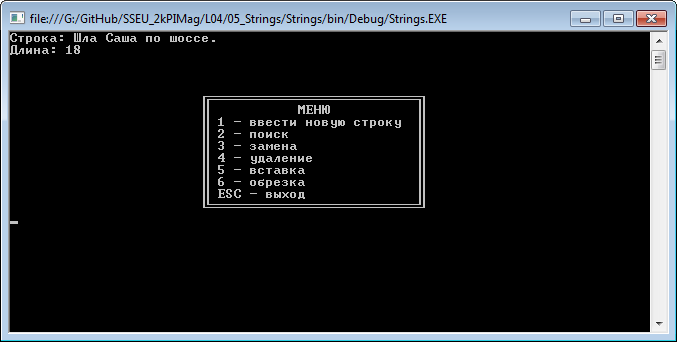
* Что будет, если пользователь введет пустую подстроку?

Для вставки подстроки (кнопка 5) потребуется ввести саму подстроку и после какого символа ее необходимо вставить.

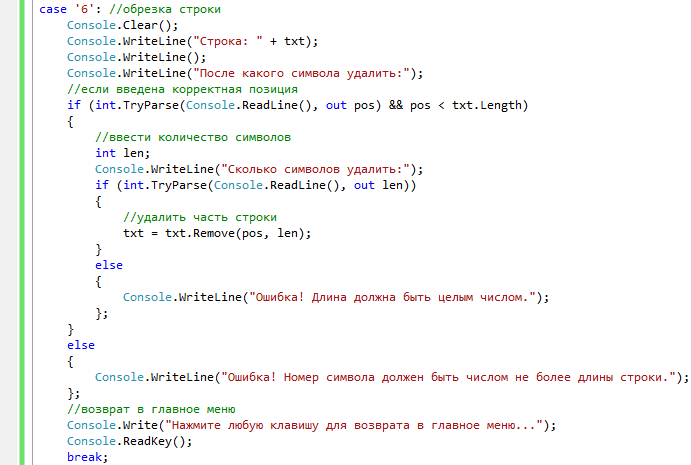


* Какой проверки не хватает в if?





Последнее действие – **обрезка строки (кнопка 6)**. Воспользуемся методом Remove. Пользователь должен ввести два числа: с какого и сколько символов нужно удалить.



* Что будет, если ввести len больше длины строки?

Проверьте работу программы по всем кнопкам.

Выполните коммит получившейся программы с комментарием «Л4.П4 Консольный интерфейс для обработки строк».

### Задание на самостоятельную работу

#### Задание 1

Напишите программу, которая для любой нажатой клавиши выводит ее код в таблице Unicode.

#### Задание 2

Проверьте, является ли введенная строка палиндромом (перевертышем). Регистр символов и пробелы игнорируются.

Примеры палиндромов: “ара”, “Цыц”, “А роза упала на лапу Азора”, “123321”.

#### Задание 3

Пользователь вводит предложение. Найти самое длинное слово, вывести его и сколько в нем символов. Если слов одинаковой длины несколько, то достаточно вывести только первое из них. Слова отделяются друг от друга пробелами и знаками препинания (. , ! ? ; : -).

Например, предложение: Поиск слова, которое длиннее всех в предложении.

Результат: предложении (11)

#### Задание 4

Реализуйте метод сжатия строки на основе счетчика повторяющихся символов. Например, строка “aabcccccaaa” должна превратиться в “а2b1с5аЗ”.

*Дополнительное задание.* Если «сжатая» строка оказывается длиннее исходной, метод должен вернуть исходную строку.