regEx – regular expressions – начин на управление на информацията в стрингове. То е нещо като език

String е клас. String-овете са immutable – read only – като масивите, реално е стринг от chars. Ако искаме да редактираме създаваме нов стринг! Всички операции със стринга създават нов стринтг...и затова + оператора е празен.

String.Compare – сравняваме 2 стринга по азбучен ред. Сравнява се буква по буква. Числото, което стои зад character-а. Освен това при еднакви начални букви по-късия е по-малък. Първо буква по буква до момента в който има разлика, ако няма такава – по дължина.

Int result = string.Compare(str1, str2, true) - -1, 0 и 1

Result == 0 if str1 equals str2

-1 str 1 is before str 2, по-малък е

True не проверява за големи и малки букви, case insensitive, false проверява и сравнява и големината

== може също да се използва за case sensitive

Str1.Equals(str2) върши същата работа, но САМО ЗА C#, вика .Equals

В C# компилатора оптимизира и ако види 2 еднакви стринга ги записва на 1 място в паметта. Затова в др езици по-правилния вариант е Equals.

В други езици има и друг операто === - точно сравняване по стойност, като .Equals

Strint str = string.Concat(str1, str2);

IndexOf – намира позицията на стринг в друг стринг

Int pos = str1.IndexOf(“g”) – къде се намира g в стринга. Ако се повтаря няколко пъти взима първия.

Може и със sunstring – IndexOf(“gel”)

Ако символа го няма дава минус 1

Str2.LastIndexOf – хваща последното място, където го има

Int Index = input.IndexOf(patter, index + 1) – търси от позиция индекс + 1

Substring – част от големия стринг

Str.Substring( int startIndex, int length) – почва от начален индекс и взима определен брой символи

Може и само (int startIndex) – от позицията до края

String.Split реално взима масив от чарове, дори и да не сме ги задали изрично като new char[], но ако искаме да ползваме stringSplitOptions трябва да укажем масива, за да знае програмата до къде е масива и къде са другите опции. Най-правилно е изрично да си създадем масива преди това, да си изнесем символите в нова променлива.

String.Replace(string match, string term) – хваща стринг и го замества с друг. НЯМА ЗНАЧЕНИЕ ОТ ДЪЛЖИНАТА! АКО не намери текса НЕ ГЪРМИ, просто не прави нищо. Затова в задачата нямаме нужда от if проверката!!!!

Str.Remove( int index, int length) – работи като .substring, но премахва

С new string при дължина 0 – гърми! Затова, също, трябва да махаме празните стрингове

Str.Trim() – маха празните места! Без никакви параметри. САМО празните места. Може да работи по стринг от елементи.

Имаме и TrimStart и TrimEnd

String.Join(“”,string.Reverse())…. – така може да обърнем стринг. Може и с new string

String buildera има отделена памет за буфер, в която може да се пълни например, без да презаписва. Ако преминем паметта се създава нов с двоен капацитет

Той е отделен клас

има Capacity – капацитета, а не конкретната дължина, по default e 16

new StringBuilder(int) – указваме изрично капацитет

.Append – добавя. С него може да си направим метод reversestring

StringBuilder sb = new String.Builder();

For I = length -1, 0, --

Sb.Append(s[i])

System.Diagnostics има е хронометър – Stopwatch st = new Stopwatch(),

St.Start; st.Stop; st.Elapsed.

Regular Expressions – regEx – примерно регулярен израз за emai; - [a@a.a](mailto:a@a.a)

[---@---.---](mailto:---@---.---)

Това реално си е отделен език. Той реално ни трябва навсякъде.

Използва се върху стринг! Търси информация в стринг

Regexr.com

[] – намира дадени символи

[^]-всичко, което не е това

[\*-\*] от до – [0-9]

\w – a-zA-Z0-9\_ - word

\W – всичко, което не е \w

\d – числа

\D – всичко, което не е числа

\s – табулация, празно място, нов ред

[a-z]+ - всички части от думата, които са малки букви.

\*- може да го има може да го няма

\t\w+ - табулация, име с 1 или повече символа

\d+ - число един или повече пъти

+ - една или повече

Котви – началото и край на нещо дълго, примерно стринг

^[a-z] – хваща символа в началото

^[a-z] + -хваща първата дума

$ - хваща само края

\b – word boundary – от тук започва думата, търсим го като цяла дума

{} – определен брой пъти

\d{2} – дума два пъти

() – група

(!) – група, която няма да пази като група

| - или

\1 – reference – връща вече създадена група

System.Text.ReqularExpressions

String pattern = @”A\w+”;

Regex regex = new Regex(pattern);

IsMatch (string text)

Bool = regex.IsMatch(text);

Regex.Match

Match match = regex.Match(text); - имаме индекса на текста, дължината му

Cw(match) – принтира първия мач

MatchCollection – взима всички мачове, прави масив от мачове

Cw(matches[0], matches[1])

Има групи ако ползваме скоби в регекса. 0 ни е целия мач, после имаме отделните групи

match.Groups[1]

--

Foreach с match collection даваме Match нещо си, а не var нещо, защото не знае какво да вземе

Regex.Replace(text, replacement) – прави нов стринг, запаметява в него и после маха стария

. означава каквото и да е в регекса

\*? – хваща възможно най-малко неща, lazy – нула или повече пъти, като намери 0 – спира и мачва

Адрес може да хванем с “\S” – между кавимките няма празно място

--

Regex split – можем да сплитваме и по регекс