**Stringy, regulární výrazy, parsování textu, kódování**

**Stringy**

String je objekt, který je uložen na haldě a reprezentován jako pole charů, které jsou v c# jako UTF-16.

Díky jeho struktuře v poli charů, obsahuje metody využívající indexy, třeba na získání písmena na indexu nebo smazání písmen od indexu,… také se dá rozdělovat pole stringů nebo charů

String je jediný objekt v C# u kterého se při vytvoření nového nepoužívá slovo “new“. Toto slovo se používá, pokud by bylo v zájmu vývojáře udělat nový stejný objekt, protože pokud by se nepoužilo a dal by se to stringu text, který je již jako string objekt na heapu, tak by byl odkázán na tento. Pokud by bylo new, tak je odkázán na svůj vlastní.

Pokud se obsahu string upraví, tak se na heapu vytvoří úplně nový objekt s novým referencí na nové pole charů. To by mohlo být problém při iterace nějaké kolekce, protože by to bylo pomalé, kvůli vytvářením nových obejetků typu string a alokaci paměti pro jejich vytvoření. A také by zabírali větší místo na haldě, kterou by musel uklízet Garbage Collector. Proto pro iteraci by se měl použít StringBuilder, který nevytvoří nový objekt, ale pouze přidáváním modifikuje stejný objekt. Ten s tvorbou polí charů pracuje, tak, že vytvoří pole o velikosti 16 a poté další, které se napojeno na to staré, pokud bude tedy velikost 16+16 charů překonána, tak vytvoří vetší pole o velikosti 32. Funguje tedy jako spojový seznam polí. A neprovádí alokaci a překopírování za účelem zvětšení kapacity.

**Parsování**

Parsováním je převod z textové podoby na nějaký specifický typ, základní: čísla, čísla s čárkou a pokročilé jako IP Adresa.

Mnoho datových typů (těch základních) má v C# již připraveny metody k parsování.

Pokud se parsování nezdaří, tak bude vyhozena FormatException, proto by se mělo parsování nějak hlídat, třeba regexem nebo je často dostupná metoda TryParse() která vrací hodnotu bool, a out datový typ převedený, pokud nebude převeden, tak má svojí základní hodnotu.

**Kódování**

Text, jehož znaky jsou zpracovávány jako číselné hodnoty.

Kódování textu přiřazuje k znakům tyto číselné hodnoty prostřednictvím tzv. znakových sad

Znakové sady je často liší velikostí.

**ASCII (American Standard Code for Information Interchange) 7-bit**

Definuje 128 znaků (0-127)

Obsahuje znaky anglické abecedy…

Kvůli absenci znaků s diakritikou byla tato znaková sada rozšířena na 8bitovou, které umožňuje až 256 znaků.

**Windows-1250 8-bit**

Je výchozí znakovou sadou pro kódování češtiny v systému MS Windows.

Toto kódování lze používat nejen pro češtinu, ale i pro další středoevropské jazyky (albánština, chorvatština, polština, slovenština a další) a pro němčinu.

**ISO 8859-2**

Podobná Windows 1250, liší se pozice znaků. Je používaná na linuxových systémech.

**Unicode – znaková sada.**

Tato znaková sada by měla obsahovat všechny znaky z používaných abeced různých jazyků.

Je implementována jako UTF-8, UTF-16, UTF-32

Výhodou je, že vyšší utfka obsahují znaky většiny jazyků, nevýhodou je vyšší velikost bitu na znak.

**Regulární výrazy**

Regulární výrazy vznikly z důvodu potřeby práce s textovými řetězci určitým unifikovaným způsobem.

Poskytují výkonný, flexibilní a efektivní způsob zpracování textu. Rozsáhlý zápis vzorů regulárních výrazů, který odpovídá vzoru, vám umožní rychle analyzovat velké objemy textu na:

* Hledání specifických vzorů znaků.
* Ověřte text, abyste se ujistili, že odpovídá předdefinovanému vzoru, tomuto se říká validace textu.
* Umožňují vyhledávání podřetězců v řetězci, jednodušším způsobem než pomocí podmínek, Také umožňují úpravu nebo nahrazování znaků.

Pro mnoho aplikací, které pracují s řetězci nebo které analyzují velké textové bloky, jsou regulární výrazy nepostradatelným nástrojem.

Regulární výraz je textový řetězec složený z určitých znaků.

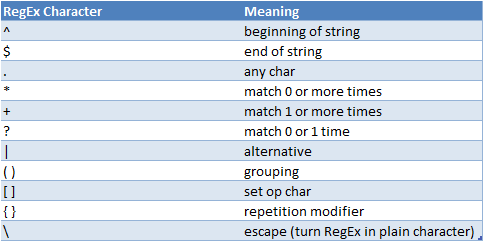
Gramatika regulárních výrazů je poměrně nepřehledná a proto je dobré již napsané výrazy komentovat.

**Třída Regex**

Umožňuje pracování s regulárními výrazy.

Do jejího konstruktoru se dává regulární výraz, Poté se pomocí metody IsMatch(string ke kontrole) ověří, jestli string splňuje regulární výraz.

**Zápis regulárních výrazů**



Tečka

libovolný znak (jeden), pokud budou třeba 3 tečky tak 3 libovolné znaky

 projde

projde

Protože výraz byl aplikován na počet prvkjů které potřeboval, další nekontroluje.

To se řeší pomocí začátku a konce výrazu. ^ = začátek, $ = konec

 neprojde

Hranaté závorky

slouží pro definici znaků, které řetězec smí nebo nesmí obsahovat, pokud se nesmí tak se před první znak v závorce dá ^

Kulaté závorky

Backslash

Umožňuje použití nějakého znaku, který má roli v regulárních výrazech třeba ^ aby byl použit jako část výrazu, kterou by měl string splňovat.

Složené závorky - kvantita

Udávají počet kolikrát je má daný znak/znaky opakovat.

{počet} – počet opakování, {min,max} – rozmezí opakování.

? – {0,1}

\* - {0, nekonečno}

+ - {1, nekonečno}

Také je definováno několik písmen pro skupiny znaků.

