Práce s textovými řežezci

string (== class System.String)

class StringBuilder

class Regex

string

Zpracování textu

- Čtení textových souborů
- Hledání klíčových slov
- Validace vstupů
- Analýza textů

Manipulace s textem

- Srovnávání
- Vyhledávání
- Výběr
- Rozdělení podle separátorů

Co je string

- Sekvence znaků, která je uložená na určité adrese v paměti
- Char = jeden znak
- .NET používá **Unicode** (na rozdíl od starší ASCII 8 bitové tabulky)

  Unicode je 16 bitový => můžeme přímo uložit 2^16 = 65 536 znaků + některé znaky se skládají ze dvou kódů

  dohromady můžeme uložit 100 000+ znaků

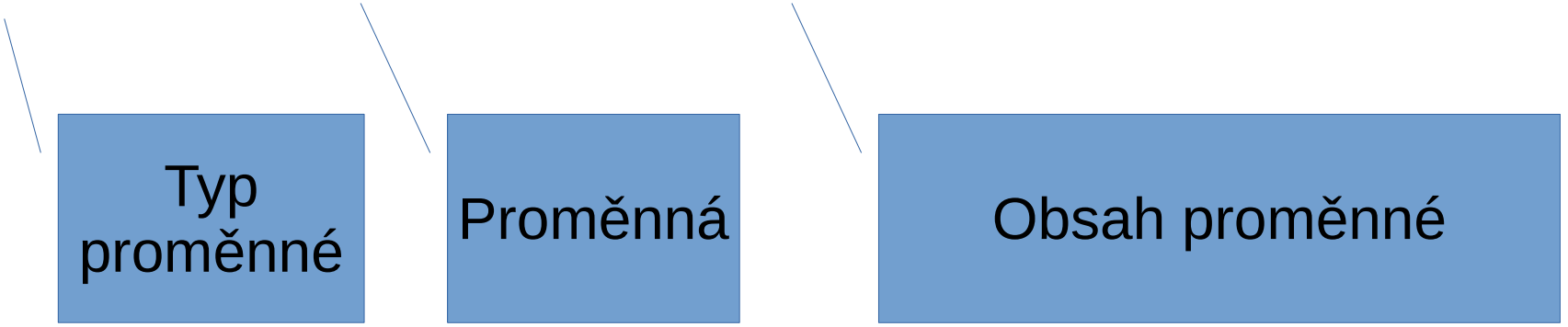
  V jedné tabulce znaky pro všechny jazyky + symboly

Class System.String

System.String = string (C# alias)

Deklarace stringu:

```
string uvitani = "Hello, C#";
```



Class System.String

V paměti:

```
string uvitani = "Hello, C#";
```

H	e	l	l	o	,		C	#

Interní reprezentace = pole znaků (`char[]`)

String je **třída** a je dynamicky alokovaná na heapu (haldě)

Proměnná typu string obsahuje **ukazatel** na objekt => string je referenční typ

Class System.String

- String má důležitou vlastnost:

  Je neměnitelný(!) - anglicky „immutable“

  Jakmile je jednou inicializovaný, jeho obsah se nemůže měnit.

- Pokud potřebujeme měnitelný string, je nutné použít třídu **StringBuilder** (viz dále).

StringExample1

Class System.String

Se stringem můžeme zacházet jako s polem

```
string str = "abcde";
char ch = str[1]; // ch == 'b'
str[1] = 'a'; // Compilation error!
ch = str[50]; // IndexOutOfRangeException
```

Uvozovky nejsou součástí stringu!

Class System.String

Stejně jako pole má string vlastnost Length:

```
string message = "This is a sample string.";
for (int i = 0; i < message.Length; i++)
{
   Console.WriteLine("message[{0}] = {1}", i,
      message[i]);
}
```

Class System.String

Deklarace proměnné `str` typu `string`:

```
string str;
```

Nealokuje prostor pro string, je to jen informace pro kompilátor, že s proměnnou `str` se bude zacházet jako se stringem. Hodnota je **až do přiřazení** `null` (žádná hodnota) a nemůžeme s ní zatím pracovat.

Inicializace stringu:

```
str = "Kniha se jmenuje \"Kdo chytá v žitě\"";
```

Escaping

Escaping

Vyzkoušejte

Class System.String

Inicializace stringu:

- Přiřazením „literálu“ - konstanty

```
str = "Kniha se jmenuje \"Kdo chytá v žitě\"";
```

- Přiřazením hodnoty jiného stringu

```
string s = str; // s a str ukazují na stejné
                // místo v paměti
```

- Operací, která vrací string

```
string s = 2.ToString ();
string s = "Kniha se jmenuje " + "\"Hrozny hněvu\".";
```

Operace s řetězci (výběr)

Test na shodu

Metoda Equals a operátor == funguje stejně:


```
string word1 = "C#";
string word2 = "c#";
Console.WriteLine(word1.Equals("C#"));    // True
Console.WriteLine(word1.Equals(word2));   // False
Console.WriteLine(word1 == "C#");    // True
Console.WriteLine(word1 == word2);   // False
```

Operace s řetězci (výběr)

Abecední pořadí

operátory < a > nefungují s řetězci, namísto toho existuje metoda CompareTo(...), která vrací -1, 0 nebo 1 podle abecedního pořadí porovnávaných řetězců:

-1 první string je abecedně před druhým stringem

0 stringy jsou totožné

1 první string je abecedně za druhým stringem

Operace s řetězci (výběr)

Metoda CompareTo (string s)


```
string score = "sCore";
string scary = "scary";
Console.WriteLine(score.CompareTo(scary));
Console.WriteLine(scary.CompareTo(score));
Console.WriteLine(scary.CompareTo(scary));
```

Operace s řetězci (výběr)

Varování:

Abecední pořadí nesouvisí s pořadím písmen v tabulce Unicode, malá písmena jsou abecedně před velkými.

'a' je před 'A'

přestože v Unicode má 'A' kód 65 a 'a' kód 97 !

Další zvláštnosti má použití jazykových specifik.

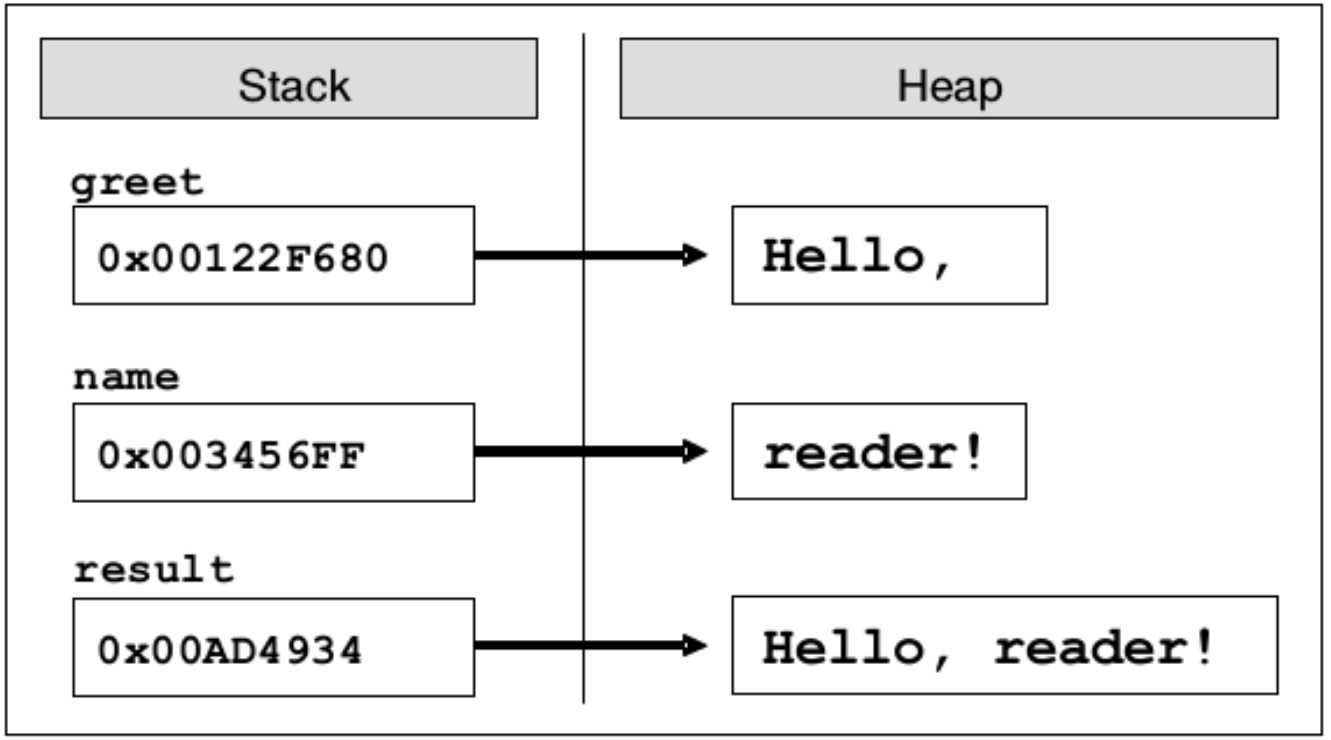
StringExample2

Operace s řetězci (výběr)

Spojování řetězců

```
string greet = "Hello, ";
string name = "reader!";
string result = greet + name;
```

StringExample1



Operace s řetězci (výběr)

Převádění na malá/velká písmena

```
string greet = "Nazdar, ";
string name = "Světe!";
string result = greet + name;
```

- `Console.WriteLine (result.ToUpper());`
- `Console.WriteLine (result.ToLower());`

StringExample3

Operace s řetězci (výběr)

Vyhledávání řetězce v jiném řetězci

```
int i = 5;
string s = "Číslo " + i + " žije.";

Console.WriteLine( s.IndexOf("žije")); // 8
```

IndexOf vrací:

-1 pokud se hledaný řetězec v prohledávaném řetězci nenachází

i > 0, pokud je hledaný řetězec nalezen na pozici i

StringExample3

Operace s řetězci (výběr)

Opakované vyhledávání řetězce v jiném řetězci

```
string quote = "The main intent of the \"Intro C#\"" +
   " book is to introduce the C# programming to newbies.";
string keyword = "C#";


int index = quote.IndexOf(keyword);
while (index != -1)
{
   Console.WriteLine("{0} found at index: {1}", keyword, index);
   index = quote.IndexOf(keyword, index + 1);
}
```

= Index počátku dalšího vyhledávání

Operace s řetězci (výběr)

Výběr části řetězce

Metoda `Substring ( startIndex, length )` vybere ze zadané instance řetězce tu část, která začíná na pozici `startIndex` a má délku `length`.

```
s = "Číslo 5 žije. A číslo 6 už nežije?"

Console.WriteLine ("".PadLeft(16) + s.Substring (16, 5));

// 0123456789012345678901234567890123
// Číslo 5 žije. A číslo 6 už nežije?
//                 číslo
```

16 pozic
(od 0 do 15)

5 písmen od 16 pozice

StringExample3

Operace s řetězci (výběr)

Rozbití řetězce na pole

Metoda `Split ( separators )`
rozbije řetězec na části rozdělené separátory a vrátí pole řetězců `string[].`

```
string nonsense = "Vidličky a nože běhají po dvoře.";
char[] separators = new char[] {' ', '.'};
string[] nonsep = nonsense.Split (separators);

for ( int j = 0; j < nonsep.Length; j++
   Console.WriteLine ( j + " : " +nonsep[j]);
```

- Další praktická verze metody Split nabízí možnost eliminovat prázdné řetězce ve výsledném poli:

```
string[] nonsep = nonsense.Split(separators,
                  StringSplitOptions.RemoveEmptyEntries);
```

StringExample3

Operace s řetězci (výběr)

Nahrazení řetězce v řetězci

Metoda `Replace (string staraH, string novaH)`

nahradí ve stringu všechny výskyty (přesné) prvního argumentu druhým argumentem.

```
string s = "Oxid uhličitý je nejběžnější mezi oxidy";

s = s.Replace ("Oxid", "Kysličník");

Console.WriteLine(s);

// Kysličník uhličitý je nejběžnější mezi oxidy
```

I v tomto případě se vytvoří nový string, ve kterém jsou provedeny změny
(string je neměnný!)

Operace s řetězci (výběr)

Formátování řetězce

Metoda `ToString (...)`
Každý objekt v .NET má metodu `ToString`, která poskytuje textovou reprezentaci objektu. Metoda je volaná vždy, pokud vypisujeme pomocí `Console.WriteLine`.

```
DateTime currentDate = DateTime.Now;

Console.WriteLine(currentDate);

// Výstup: 01.02.2012 13:34:27 podle lokálního nastavení,
// „culture settings“
```

Interně se volá metoda DateTime.ToString()

StringExample3

Operace s řetězci (výběr)

Formátování řetězce

Metoda `String.Format(...)`
`Console.WriteLine` používá formátovací string s parametry („placeholders“)

```
Console.WriteLine("This is a template from {0}","David");
```

Před tím, než se vytiskne string, doplní se do formátovacího stringu argumenty v pořadí, v jakém jsou posílány do metody WriteLine.
První argument za formátovacím řetězcem se doplní místo zástupného symbolu {0}, druhý do {1}, atd. Počet zástupných symbolů musí být stejný, jako je počet argumentů po formátovacím řetězci.

```
string formattedText = String.Format(
   "Today is {0:MM/dd/yyyy} and {1} is working on {2} in {3}.",
   date, name, task, location);
Console.WriteLine(formattedText);
```

4 zástupné symboly
({0:MM/dd/yyyy},{1}, {2}, {3})
4 argumenty
(date, name, task, location)

Operace s řetězci (výběr)

Interpolace řetězce

```
static void Main(string[] args)
{
    DateTime today = DateTime.Now;
    int n = today.DayOfYear;

    string s = $"Dnes je {today:d/MM/yyyy} a to je {n}. den v roce.";

    Console.WriteLine(s);
}
```

Znak $ povinně před každým interpolovaným stringem

Příklad

Napište program který určí, kolikrát se daný řetězec nachází v textu. Otestujte na libovolném textu.

Např. Kolikrát se nachází v textu „He's a real nowhere man, sitting in his nowhere land, making all his nowhere plans for nobody.“ řetězec „nowhere“? (3)

Příklad

Napište program který velkými písmeny napíše část textu uzavřenou značkami <upcase> a </upcase>. Otestujte na libovolném textu.

Např. „He's a <upcase>real nowhere man</upcase>, sitting in his nowhere land, making all his nowhere plans for nobody.“

(„He's a REAL NOWHERE MAN, sitting in his nowhere land, making all his nowhere plans for nobody.“)

Class StringBuilder

Co je špatného na obyčejném stringu?

```
string str1 = "Super";
string str2 = "Star";
string result = str1 + str2;
```

1. Vytvoření nového prostoru pro součet na heap - pomalé
2. Spojení str1 a str2, uložení na heapu - pomalé
3. uložení adresy výsledku do result

Nezměnitelnost stringu má výhody z hlediska stability kódu, ale z hlediska výkonnosti je omezující.

Class StringBuilder

Co je špatného na obyčejném stringu?

```
string collector = "Numbers: ";
for (int index = 1; index <= 200000; index++)
{
   collector += index;
}
```

V každém průběhu cyklem se vytváří nová proměnná typu string.
Kopíruje se obsah staré proměnné collector.
Nepoužité proměnné odstraňuje *garbage collector*.
Doba běhu řádově v minutách, **podobnému scénáři je dobré se vyhnout!**

StringExample4

Class StringBuilder

Použití StringBuilder:


```
StringBuilder sb = new StringBuilder();
sb.Append("Numbers: ");

for (int index = 1; index <= 200000; index++)
{
   sb.Append(index);
}
```

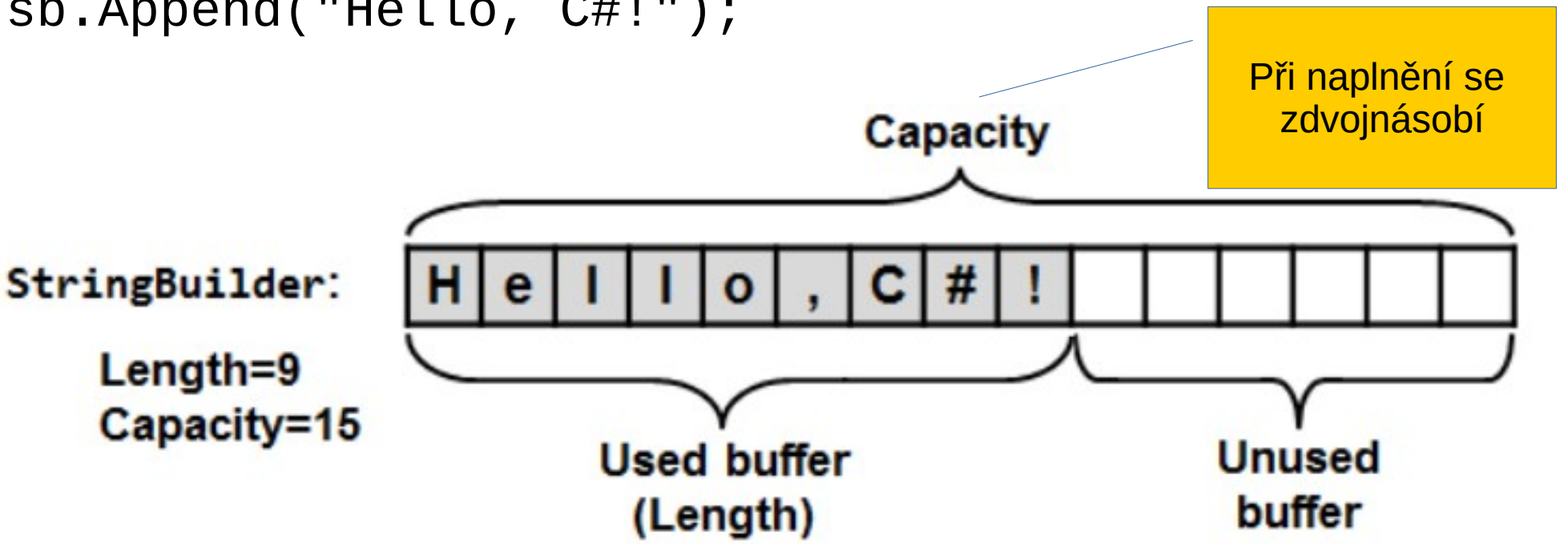
StringExample4

Paměť se alokuje v blocích podle toho, jak narůstá velikost sb, vše se odehrává na jedné kopii dat. Při změně dat nedochází k vytvoření nové proměnné typu string.
Garbage collector nemusí odstraňovat staré proměnné.
StrinBuilder je změnitelný, nepřesouvají se žádné bloky paměti.
Doba běhu řádově v desetinách vteřin.

Class StringBuilder

Struktura StringBuilder

```
StringBuilder sb = new StringBuilder(15);
sb.Append("Hello, C#!");
```



StringExample4

Class StringBuilder

Konstruktor

Namespace `System.Text`

`StringBuilder(int capacity)`

pokud víme, jak může být objem dat veliký, inicializace zrychlí výpočet (omezí se navyšování kapacity po vyčerpání bufferu)

Class StringBuilder

Metody

`Append(...)` přidává na konec bufferu

`Clear(...)` zruší celý buffer

`Remove (int startIndex, int lenght)`
vymaže buffer od pozice startIndex délky length

`Insert (int offset, string str)`
vloží string na pozici offset

`Replace (string oldValue, string newValue)`
nahradí všechny výskyty oldValue za newValue

`ToString()` vrátí buffer jako string

StringExample4

Příklad

Napište program, který pomocí StringBuilder nahradí část textu uzavřenou značkou <upcase> na <UPCASE> a </upcase> za </UPCASE>. Otestujte na libovolném textu.

Např. „He's a <upcase>real nowhere man</upcase>, sitting in his nowhere land, making all his nowhere plans for nobody.“ změní na:
„He's a <UPCASE>real nowhere man</UPCASE>, sitting in his nowhere land, making all his nowhere plans for nobody.“

Regulární výrazy

Typické úlohy (na co to je - příklady):

- Projdi všechny stránky na svém webu a změň zastaralou značku a za a
- Projdi knihu a najdi všechna místa, kde se objevují jména postav Vinnetou nebo Old Shatterhand
- Najdi ve všech souborech na disku kombinaci svého jména a příjmení

Regulární výrazy jsou silným nástrojem pro zpracování textu pomocí **vzorů**.

Prohledávání, filtrování a nahrazování textů, hledání emailových adres, validace dat.

Regulární výrazy

Regulární výrazy znáte, ale možná o tom nevíte:

DOS command line:

```
dir *.txt           // 1.txt, uvod.txt, cokoliv.txt
dir a?.txt          // a.txt, a1.txt, aa.txt
dir a*1.txt         // aaaa1.txt
del *.*             // vymaž vše v adresáři
```

Linux:

```
ls a[12].txt        // a1.txt, a2.txt, ne a.txt
ls a?.txt           // a1.txt, a2.txt, aa.txt
rm ./*.*            // vymaž vše v adresáři
```

Regulární výrazy

Potřebujete:

Vzorek textu

Znát konstrukci regulárního výrazu

Znát třídu Regex

Regulární výrazy

Vzorek textu

Vyberte si vzorek textu, který je charakteristický pro vaši úlohu.

Úkol	Text
Formátuj data ze vstupního souboru	2.2.1980 20:00:00 2. 2.1980 20:00:00 2.2. 1980 20:00:00 2. 2.1980 20:00 2.2.1980 20:00:00
Hledej a nahrazuj string z webové stránky	<HR> <CENTER><SMALL> <A HREF="#podrobnosti">Podrobnosti</A> | <A HREF="#software">Dostupný software</A> | <A HREF="#hardware">Požadovaný hardware</A> | <A HREF="#odkazy">Odkazy na další informace</A> </SMALL></CENTER> <HR>

Regulární výrazy

Znát konstrukci regulárního výrazu

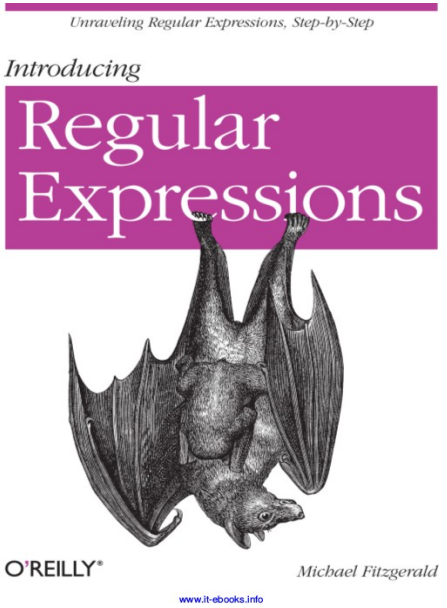
Ideální prostředí pro vyzkoušení:

https://regex101.com

http://regexpal.com

http://www.regexr.com

152 stran.
A to je jen „úvod“!



Varování: je to složité a nedá se to naučit jinak než metodou pokus - omyl!

Regulární výrazy

Znát konstrukci regulárního výrazu

Znaky

`x`	znak x (platí pouze pro alfanumerické znaky)
`\`	vrací původní význam znaku
	(např \\ zpětné lomítko, * hvězdička)
`.`	libovolný znak
`\0n`	znak v osmičkovém kódu 0n (0 <= n <= 7)
`\x`*`hh`*	znak v hexadecimálním kódu 0xhh
`\u`*`hhhh`*	znak unicode v hexadecimálním kódu 0xhhhh
`\t`	znak tabulátor ('\u0009')
`\n`	znak nového řádku ('\u000A')
`\r`	znak posunu vozíku ('\u000D')

Regulární výrazy

Znát konstrukci regulárního výrazu

Skupiny znaků

`[xyz]`	*x*, *y* nebo *z*
`[^xyz]`	všechny znaky kromě *x*,*y* a *z*
`[a-z]`	*a* až *z*
`[a-f[k-r]]`	*a* až *f* nebo *k* až *r* jinak: `[a-fk-r]`
`[a-z&&[^k-o]]`	*a* až *z* kromě *k*, *l*, *m*, *n* a *o*

Regulární výrazy

Znát konstrukci regulárního výrazu

Skupiny znaků

`\d`	číslice [0-9]
`\D`	opak číslice [^0-9]
`\s`	bílý znak: [\t\n\x0B\f\r]
`\S`	opak bílého znaku [^\s]
`\w`	slovo - alfanumerické znaky [a-zA-Z_0-9]
`\W`	opak slova [^\w]

Regulární výrazy

Znát konstrukci regulárního výrazu

Kvantifikátory (kolik, kolikrát?)

`?`	žádný nebo jeden
`*`	žádný nebo více
`+`	jednou nebo více
`{n}`	právě n-krát
`{m,n}`	minimálně m-krát, maximálně n-krát
`{n,}`	minimálně n-krát

Regulární výrazy

Znát konstrukci regulárního výrazu

Ukotvení (kde ve stringu?)

^	začátek stringu
$	konec stringu
\b	hranice slova
\B	mimo hranici slova

Regulární výrazy - příklad

Vzorek textu

Vstupní text:

```
Operační systém Linux je volně šiřitelný OS
typu UNIX. Jeho autorem je Linus Torvalds a
mnoho dalších programátorů v Internetu. Jádro
Linuxu je volně šiřitelné (public domain)
podle pravidel GNU General Public License
(licence používaná u softwaru GNU). Linux byl
původně psán pro architekturu IBM PC s
procesorem i386 a vyšším. V současné době
existují i verze pro m68000, MIPS, Sun Sparc,
DEC Alpha/AXP a některé další architektury.
Jednou z hlavních výhod oproti komerčním
UN*Xům je jeho nulová cena, dále snadno
dostupný základní software a v neposlední řadě
také nízké nároky na hardware a velmi příznivý
výkon. Často je také k dispozici více
dokumentace než k jiným systémům.
```

RegexText.txt

Regulární výrazy - příklad

Znát konstrukci regulárního výrazu

Shody (Match):

Regulární výraz	Nalezeno
t[aeiou]r nebo t.r nebo t[oeu]r	autorem, programátorů, Internetu, architekturu, některé, architektury
\bU.+X nebo \bU..X	UNIX, UN*Xům
x.*e	Linux je, Linuxu je, Linux byl původně psán pro architekturu existují i verze
[Oo][a-zA-Zá-ž]* ^[Oo][a-zA-Zá-ž]*	Operační, volně, OS Jeho, autorem, Torvalds, programátorů, Jádro, volně Operační

Regulární výrazy - příklad

Znát konstrukci regulárního výrazu

Najděte regulární výrazy, které najdou v dříve uvedeném textu slova uvedená v pravém sloupci tabulky. Použijte https://regex101.com

Regulární výraz	Nalezeno
?	Linux
?[Ll]icen[sc]e	License, licence
?	Linux, Linus, Linuxu
?Linux [je|byl]{1,}	Linux je, Linux byl

Vyzkoušejte

RegexText.txt

Regulární výrazy obecně

Znát konstrukci regulárního výrazu

Začínáme

Vzor	Prohledávaný text = „Regular expressions“
.	Vyhovuje R, e, g, u, l, a, r, , e, x, p, r, e, s, s, i, o, n, s
..	Vyhovuje Re, gu, la, r , ex, pr, es, si, on
.+	Vyhovuje při „ungreedy“ R, e, g, u, l, a, r, , e, x, p, r, e, s, s, i, o, n, s
.+	Vyhovuje při „greedy“ Regular expression
.*	Vyhovuje při „ungreedy“ ", R, ", e, ", g, ", u, ", l, ", a, ", r, ", ...
.*	Vyhovuje při „greedy“ Regular expression, "

Regulární výrazy

Znát konstrukci regulárního výrazu

Pokračujeme

Vzor	Prohledávaný text = „Regular expressions“
.?	Vyhovuje při „ungreedy“ ", R, ", e, ", g, ", u, ", l, ", a, ", r, ", ...
.?	Vyhovuje při „greedy“ R, e, g, u, l, a, r, , e, x, p, r, e, s, s, i, o, n, s, "
expres	Regular expressions
expres.*	Regular expressions
expres.+	Regular expressions
gula.+	Regular expressions (greedy = vyber nejvíc)
gula.+	Regular expressions (ungreedy = vyber nejmíň)

Regulární výrazy

Znát konstrukci regulárního výrazu

Shlukování

Vzor	Text = „Passion for regular expressions“

(?'doubles'ss)(?'on'ion)	Pojmenovaná shoda 2x:
	`01234567890123456789012345678 90`
	`Passion for regular expressions`
	
	1. doubles [2, 4] = 'ss'; on [4, 7] = 'ion'
	2. doubles [25, 27] = 'ss'; on [27,30] = 'ion'

(ss)(ion)	Nepojmenovaná shoda 2x:
	`01234567890123456789012345678 90`
	`Passion for regular expressions`
	
	1. $1 [2, 4] = 'ss'; $2 [4, 7] = 'ion'
	2. $2 [25, 27] = 'ss'; $2 [27,30] = 'ion'
	
	Systém přiřadí shodám jména $1 a $2

Složitější regulární výrazy (hloupé)

Datum: (\d+)\.(\d+)\.(\d+) (\d+):(\d+):(\d+)

Vyhoví: 2.2.1980 20:00:00
315.30.2 27:65:70

Nevyhoví: 2. 2.1980 20:00:00
2.2. 1980 20:00:00
2. 2.1980 20:00
2.2.1980 20:00:00

Složitější regulární výrazy (chytré)

Datum:

```
(?<day>0[1-9]|1[0-9]|2[0-9]|3[01]).
(?<month>0[1-9]|1[012]).
(?<year>[0-9]{4})
(?: (?<hour>[0-1]\d|2[0-3]):
(?<minute>[0-5]\d):
(?<second>[0-5]\d)|)
```

Vyhoví:

```
02.02.1980 20:00:00
31.02.1980 20:00:00   !
31.02.1980
```

Nevyhoví:

```
2.2.1980  20:00:00    !
2. 2.1980 20:00:00
2.2. 1980 20:00:00
2. 2.1980 20:00
2.2.1980  20:00:00
```

MATCH:

day	[0-2]	`02`
month	[3-5]	`02`
year	[6-10]	`1980`
hour	[11-13]	`20`
minute	[14-16]	`00`
second	[17-19]	`00`

Regulární výrazy

Znát třídu Regex

Namespace System.Text.RegularExpressions obsahuje statické i nestatické verze:

```
bool IsMatch(...)                    // našel? (true/false)

Match Match(...)                     // najde první

MatchCollection Matches(...)
// najde všechny Match a uloží je do MatchCollection
// foreach (Match m in Matches (…)

Replace(...)                         // nahradí všechny
```

Příklad použití: RegexExample1