Znaky

- každému znaku je nutné přiřadit nějaké číslo (nezáporné – přímé mapování znak <-> číslo) → kódování
- jiný příklad kódování → morseova abeceda
- existuje mnoho kódů → problémy
- pojmy:
  - znaková sada → které znaky budou kódovány na jaká čísla (např. A → 65)
  - charset, (kódování) → jak budou v rámci znakové sady čísla uložena (1 znaková sada = i více charsetů)
  - 65 jako 1B, 65 jako 4B, …

ASCII

- American Standard Code for Information Interchange
- = Znaková sada i charset pro základní znaky anglické abecedy, číslice, netisknutelné znaky (enter, tab apod.), jedno z nejstarších kódování
- 7-bitová (nejvyšší bit je 0) → čísla 0 – 127

	ASCII kód	
Řídící znaky	0 – 32	
číslice	0	48
	1	49
		|
	9	57
velká písmena	A	65
	B	66
		|
	Z	91
malá písmena	a	97
	b	98
		|
	z	123

- Na uložení 1 znaku → 1 B (horních 128 b nevyužito)
- Základ všech ostatních sad

8b sady

- Rozšíření ASCII
- Např.:
  - Win1250 až Win1258 – pouze ve Windows; východoevropské země = Win1250 (CP1250)
  - ISO 8859-1 až ISO 8859-15 – mezinárodní standard, starší distribuce Linuxu; čeština je ISO 8859-2
  - CP4xx, CP8xx – pro MS-DOS (čeština CP852)

  - Kameničtí, …
- Princip:
  - Dolních 128 B = ASCII
  - Horních 128 bytů → znaky národních abeced
- Problém špatně zvoleného kódování → nefunkční znaky z horní poloviny tabulky
  - text v ISO8859-2 zobrazený jako Win1250

> NYMBURK - Podle jeho posledních informací se vąak zvaľuje i verze, ľe se na muľi vznítil destilát, kterým byl polit. Muľ byl prý v silně podnapilém stavu, uvedl pro web mluvčí středočeských hasičů Ladislav Křivan.

Unicode

- www.unicode.org
- pojme všechny národní abecedy
- původně 16b (až 65 536 možných znaků), od verze 2.0 32b (používá se jen 21b)
- možné charsety:
  - UTF-8 → proměnný počet bytů (1-4)
  - UTF-16 → 16b (2B)
  - UTF-32 → 32b (4B)

    …
- základ společný s ASCII

	bitů	Písmeno A
ASCII	7	0x41
Win1250, ISO8859-2, UTF-8	8	0x41
Unicode (UTF-16)	16	0x00 41
Unicode (UTF-32)	32	0x00 00 00 41

Znaky v C#

- Vše Unicode
- datový typ `char` → 16b (UTF-16)

Znakové konstanty (literály)

- vždy uzavřeny mezi apostrofy`''`

Možnosti zápisu

- znak přímo
  `'4'`, `'a'`, `'*'`
- Unicode v hexa soustavě – obvykle pro „neviditelné“ znaky
  - `'\0x????'`
  - `'\u????'` → přednostně

```
char znak = '\u0041';   // A
```

- pozn: `\` má význam změny významu; `'\u0041'` = jinak nedovolená čtyřznaková konstanta. Konstanta uvozená `\` = **escape sekvence**

- znakové escape sekvence

	Unicode	Název	Význam
`\0`	0x0000	Nul	Nula
`\a`	0x0007	Alert (Bell)	pípnutí
`\b`	0x0008	Backspace	návrat o jeden znak zpět
`\f`	0x000C	Formfeed	nová stránka nebo obrazovka
`\n`	0x000A	Newline	přesun na začátek nového řádku
`\r`	0x000D	Carriage return	přesun na začátek aktuálního řádku
`\t`	0x0009	Horizontal tab	přesun na následující tabelační pozici
`\\`	0x005C	Backslash	obrácené lomítko
`\'`	0x0027	Single quote	apostrof
`\"`	0x0022	Double quote	uvozovky

Znaky a klávesnice

- načítání znaku:

```
char znak = Char.Parse(Console.ReadLine());
```

  - pokud zadán > 1 znak → pád programu

- čekání na stisk klávesy (klávesa nebude zobrazena na konzoli)

```
Console.Write("Pro pokračování stiskněte jakoukoli klávesu...");
Console.ReadKey();
```

Práce se znaky – ruční

- znak = číslo → lze provádět mat. operace
- implicitní konverze (uložena odpovídající Unicode hodnota) na typy:
  - všechny reálné
  - celočíselné `ushort` a vyšší
- číslo lze přetypovat na `char` → uložen znak odpovídající Unicode hodnotě

```
char znak = (char)88;   // X
```

- Příklad: převod velkého písmena na malé

```
const int offset = 'a' - 'A';       // a = 97, A = 65
char velke = 'U';
char male = (char)(velke + offset); // u
```

- Příklad: test, je-li zadaný znak číslice

```
char znak = Char.Parse(Console.ReadLine());
if ((znak >= '0') && (znak <= '9'))
{
      Console.WriteLine("je to cislice");
}
```

Práce se znaky – třída `Char`

Konverze

- `char Char.ToLower(char c)`
  pokud je `c` velké písmeno, převede jej na malé, ostatních si nevšímá

- `char Char.ToUpper(char c)` → obráceně

- Příklad: test, je-li zadaný znak `'a'` bez ohledu na velikost

```
char znak = Char.Parse(Console.ReadLine());
if (Char.ToLower(znak) == 'a')
{
      Console.WriteLine("je to a nebo A");
}
```

Testy

- Všechny metody vrací `bool`
- Číslice (0 – 9) → `IsDigit(char c)`, `IsNumber(char c)`
- Písmeno (A – Z nebo a – z) → `IsLetter(char c)`
- Písmeno nebo číslice → `IsLetterOrDigit(char c)`
- Malé písmeno → `IsLower(char c)`
- Velké písmeno → `IsUpper(char c)`
- Příklad: test, je-li zadaný znak číslice

```
char znak = Char.Parse(Console.ReadLine());
if (Char.IsDigit(znak))
{
      Console.WriteLine("je to číslice");
}
```

Řetězce

- = posloupnost Unicode znaků (`char`)
- datový typ `string` (CTS třída `System.String`)
- referenční typ

Základy práce

- deklarace proměnné, konzole

```
Console.Write("Zadej jmeno: ");
string jmeno = Console.ReadLine();
Console.WriteLine("jmeno = {0}", jmeno);
```

- Zjištění délky řetězce

```
string s3 = "ahoj";
int delka = s3.Length;  // 4
```

- `string` – **<u>při čtení</u>** se může chovat jako pole znaků (`char[]`)

```
string text = "ahoj";
for (int i = 0; i < text.Length; i++)
{
      Console.WriteLine(text[i]);
}
```

- Escape sekvence v řetězcích

```
Console.Write("Ahoj\n");      // = Console.WriteLine();
string s4 = "citace: \"šla Nanynka do zelí\", atd.";
string s5 = "C:\\Dokumenty\\soubor.txt";
```

  - Lze vypnout

```
string s6 = @"C:\Dokumenty\soubor.txt";
```

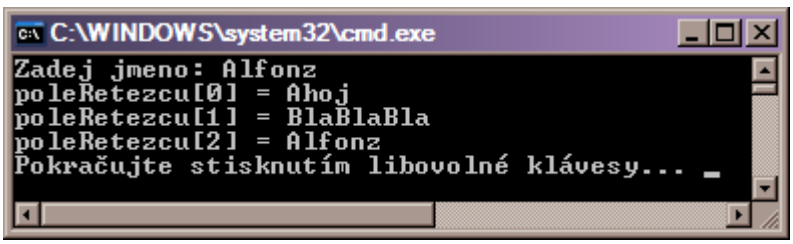
- Prázdný řetězec

```
string text1;                 // null
string text2 = "";            // reference na prazdny retezec
string text2 = String.Empty;  // reference na prazdny retezec
```

Pole řetězců

- Zcela normálně

```
string[] poleRetezcu = new string[3];
poleRetezcu[0] = "Ahoj";
string text = "BlaBlaBla";
poleRetezcu[1] = text;
Console.Write("Zadej jmeno: ");
poleRetezcu[2] = Console.ReadLine();
for (int i = 0; i < poleRetezcu.Length; i++)
{
      Console.WriteLine("poleRetezcu[{0}] = {1}", i, poleRetezcu[i]);
}
```



- Lze použít `Array.Sort()`, `Array.BinarySearch()`

Řetězce a metody

Řetězec jako parametr metody

```
static void TiskChyby(string popis)
{
      Console.WriteLine("V programu došlo k závažné chybě!!!");
      Console.WriteLine("Popis chyby: {0}", popis);
}
```

- Příklad volání

```
string error = "Dělení nulou";
TiskChyby(error);
TiskChyby("Vybuchl Vám zdroj");
```

Řetězec jako návratová hodnota

```
static string NactiString()
{
      Console.Write("Zadej text: ");
      string temp = Console.ReadLine();
      return temp;
}
```

  - Lepší řešení `return Console.ReadLine();`
- Příklad volání

```
string pozdrav = NactiString();
```

Změna řetězce uvnitř metody

```
static void InvertujVelikostPismen(ref string retezec)
{
      // u retezec zamenim mala pismena za velka a naopak
}
```

- Příklad volání

jen pro zlepšení čitelnosti kódu (`string` = referenční typ)

```
string s7 = "AhOj";
InvertujVelikostPismen(ref s7);     // s7 bude "aHoJ";
```

Práce s řetězci

- možnosti:
  - ruční
  - instanční metody
  - statické metody
  - operátory
- Instanční vs. statické metody viz další semesetr
  - instanční metody: volání ve tvaru `promenna.Metoda()`

```
string text = "ahoj";
string novy = text.ToUpper(); // "AHOJ"
```

- statické metody: volání ve tvaru `String.Metoda()`

```
string jmeno = "Martin";
string prijmeni = " Hajek";
string jmenoPrijmeni = String.Concat(jmeno, prijmeni);
```

Porovnávání

- rovnost obsahu (`==` a `!=`)

```
string s1 = "ahoj";
string s2 = "ahoj";
if (s1 == s2)
      Console.WriteLine("stejne");
if (s1 == "nazdar")
      Console.WriteLine("text1 je nazdar");
if (text1 != String.Empty)
      Console.WriteLine("text1 není prazdny");
```

- lexikografické porovnání → `int String.Compare(string s1, string s2)`
  - Vrací:
    - záporné číslo: `s1` je „menší“ než `s2`
    - 0: řetězce jsou identické
    - kladné číslo: `s1` je „větší“ než `s2`
  - Lexikografické porovnání:
    - a < b
    - ab < ac (komár < kořen)
    - a < A
    - aa < aaa (Martin < Martina)

```
int vysledek = String.Compare(text1, text2);
if (vysledek > 0)
      Console.WriteLine("text1 lexikograficky vetsi nez text2");
else if (vysledek < 0)
      Console.WriteLine("text1 lexikograficky mensi nez text2");
else
      Console.WriteLine("stejne");
```

Vyhledávání v obsahu

- Instanční metoda `bool Contains(string co)`
  - obsahuje-li řetězec podřetězec `co`

```
string osloveni = "Mile deti!";
if (osloveni.Contains("deti"))
      Console.Write("obsahuje slovo \"deti\"");
```

- Instanční metoda `bool StartsWith(string s)`
  - Test, začíná-li řetězec řetězcem `s`

```
string osloveni = "Mile deti!";
if (osloveni.StartsWith("Mile"))
      Console.Write("zacina slovem \"Mile\"");
```

- Instanční metoda `bool EndsWith(string s)`
  - Test, končí-li řetězec řetězcem `s`
- Instanční metoda `int IndexOf(něco)`
  - vyhledává první výskyt něco v řetězci
  - dle `něco` 9 přetížení:
    - co se vyhledává:
      - jiný řetězec
      - jeden znak (`char`)
    - kde vyhledávat:
      - odkud: `int startIndex`
      - kolik znaků : `int count`
  - Vrací:
    - `něco` nenalezeno: -1
    - Nalezeno: pozice prvního výskytu
  - Příklad

```
string s = "se se se: Ahoj mami, jak se máš";
int pozice = s.IndexOf("se", 8);    // = 25
```

- Instanční metoda `int LastIndexOf(něco)`
  - = `IndexOf()`, jen od konce

Úpravy

- Spojení řetězců → lze pomocí přetíženého +

```
string retezec1 = "Ahoj ";
string jmeno = "Martine!";

string pozdrav = retezec1 + jmeno;  // "Ahoj Martine!"
```

  - Jeden z argumentů číslo → automatický převod na `string`

```
string s = "a = " + 10;        // "a = 10"
```

- Instanční metoda `string Substring(int start)`
  - Vrací podřetězec od pozice `start`
- Instanční metoda `string Substring(int startIndex, int N)`
  - Vrací podřetězec od pozice `start` délky `N` znaků
  - Příklad:

```
string osloveni = "Mile deti!";
string koho = osloveni.Substring(5, 4);   // "deti"
```

- Instanční metoda `string Insert(int kam, string co)`
  - do řetězce vloží `co` od pozice `kam`
  - vrací: nově vybudovaný řetězec

- příklad:

```
string s12 = "Ahoj tati";
string s23 = s12.Insert(5, "mami a ");    // Ahoj mami a tati
```

- Instanční metoda `string Remove(int start)`
  - Vymaže znaky od pozice `start` do konce řetězce
- Instanční metoda `string Remove(int start, int N)`
  - Vymaže `N` znaků od pozice `start`

```
string osloveni = "Mile deti!";
string bezDeti = osloveni.Remove(4, 5);   // Mile!
```

- Instanční metoda `string Replace(char stary, char novy)`
  - V řetězci zamění všechny výskyty znaku `stary` za `novy`

```
string seznam = "1, 2, 3, 4, 5";
string novySeznam = seznam.Replace(',', ';'); // 1; 2; 3; 4; 5
```

- Instanční metoda `string Replace(string stary, string novy)`
  - V řetězci zamění všechny výskyty řetězce `stary` za `novy`

```
string dopis = "soudruh Novak a soudruh Hajek";
string novyDopis = dopis.Replace("soudruh", "pan");
// pan Novak a pan Hajek
```

- A mnohé další – viz. MSDN

Tisk do řetězce

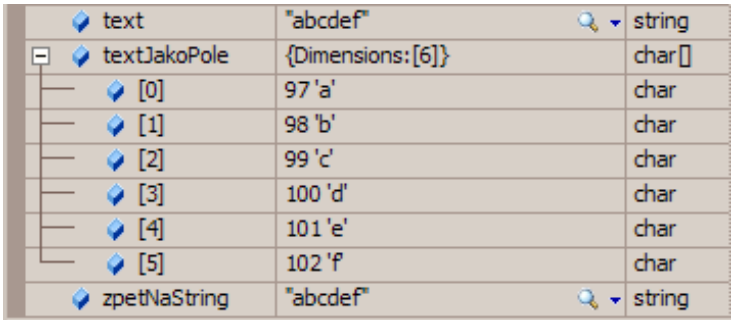
- Statická metoda `string String.Format("text s {}", vyrazy)`

```
int p1 = 10;
double p2 = 12.33;
string s = String.Format("p1 = {0}, p2 = {0:F1}", p1, p2);
// s = "p1 = 10, p2 = 12.3"
```

Převod na pole znaků a zpět

- Užitečné při složitých „ručních“ úpravách

```
string text = "abcdef";
char[] textJakoPole = text.ToCharArray();
// nejake upravy po znacich
string zpetNaString = new String(textJakoPole);
```

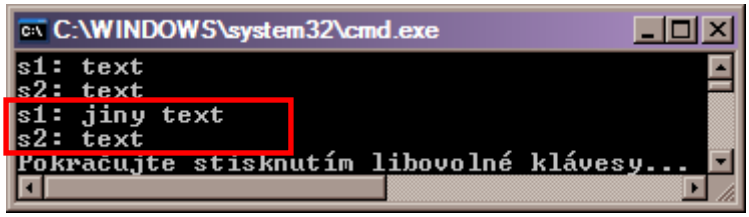


Dodatek pro pokročilé

- Řetězce jsou tzv. neměnné (immutable) → jakákoli změna v řetězci = vytvoření nového:

```
string s1 = "text";
string s2 = s1;   // = s2 odkaz na stejný objekt jako s1
Console.WriteLine("s1: {0}", s1);
Console.WriteLine("s2: {0}", s2);
s1 = "jiny text";// změna v s1
Console.WriteLine("s1: {0}", s1);
Console.WriteLine("s2: {0}", s2);
```

- vytvoření nového řetězce. Pokud na původní řetězec:
  - vede jiný odkaz (zde `s2`) → původní zůstane zachován
  - nevede jiný odkaz → bude zrušen (automatická správa paměti)



- Z toho plyne: `string` se nehodí na rozsáhlejší manipulace s textem
  - Vhodnější je `StringBuilder` nebo převod na pole znaků