Počítačové zpracování češtiny

Programovací jazyk Perl

Daniel Zeman

http://ufal.mff.cuni.cz/course/popj1/

PERL

Practical Extraction and Report Language

Praktický jazyk pro extrakci informací a jejich shrnutí

PERL

- Interpretovaný programovací jazyk pro zpracování textu.
- Snadné čtení a psaní souborů.
- Snadná práce s regulárními výrazy.
- Šetří deklarace, konverzi typů apod.
- Přenositelný: Unix, Windows i Mac.
- Nepříliš efektivní. Návrh a test v PERLu, trvalá implementace třeba v C.

Dostupnost Perlu

- Perl je zadarmo.
- Na Unixech běžně (alespoň Linux). Zkuste příkazy **`which perl`** a **`perl -v`**.
- Pro Windows ke stažení např. na:
  `http://strawberryperl.com/`
  `http://www.activestate.com/`
- Dokumentace:
  `http://www.perldoc.com/`

Spojení programů s interpretem Perl

- Unix:
  - souboru s programem umožnit spouštění:
    `chmod +x program.pl`
  - na prvním řádku programu uvést cestu k interpretu, například:
    `#!/usr/bin/perl`
- Windows:
  - dvakrát kliknout na soubor s programem, na požádání udat, že se má otvírat Perlem => **problém s předáním parametrů!**
  - při spouštění z příkazového řádku před jméno programu přidat volání Perlu, například:
    `perl program.pl`
  - v současných Windows (NT a výš) už to není (někdy!) potřeba

Skalární proměnné

```
#!/usr/bin/env perl

$a = 10;
$b = 5;
$c = $a + $b;
print "Součet ", $a, " a ", $b, " je ",
    $c, ".\n";
```

Skalární proměnné

- Proměnné se nemusí deklarovat.
- Typ je benevolentní a udává ho první znak jména: `$` znamená skalár.
- Skalár je podle kontextu řetězec, celé číslo nebo desetinné číslo.
- Do proměnných se přiřazuje pomocí operátoru `=`: `$soucet = $a + $b`.

Pole

- Proměnná začínající znakem `@` je pole.
- Není třeba předem deklarovat velikost pole.
- Prvky pole jsou skaláry. Prvky téhož pole mohou být řetězce i čísla.
- Mohou existovat dvě různé proměnné, lišící se pouze tím, že jedna je skalár a druhá pole: `$x` a `@x`. Není mezi nimi žádný vztah!

Pole

- Pozor: prvek pole je skalár, takže se k němu přistupuje takto: `$x[0]` je první (tj. nultý) prvek pole `@x`.
- `$#x` je rezervovaná proměnná: nejvyšší index v poli `@x`. Tedy poslední prvek je `$x[$#x]`.
- Alternativně: `$x[-1] == $x[$#x]`.

Příklad

```
$x[0] = "pes";
$x[1] = 34;


# $#x je původně -1, protože @x neexistuje.
$x[++$#x] = "pes";
push(@x, 34);
```

Vztah polí a řetězců

- Řetězec lze rozložit na pole skalárů (řetězců) funkcí **`split`**.
- Pole lze převést na řetězec funkcí **`join`**.
- V obou případech lze udat „oddělovač“.

Příklad

```
$avar = "abc;def;g";
@x = split(/;/, $avar);

@x = ("abc", "def", "g");
$avar = join(";", @x);
```

Řídící struktury

- Řídící struktury (podmínky, smyčky) jsou podobné jako v céčku, ale všechny bloky musí být povinně uzavřeny ve složených závorkách.
- Smyčky: **`while`**, **`for`**, **`foreach`**.
- Podmínky: **`if`** … [[**`elsif`** …] **`else`** …].

Příklad: cyklus **while**.

```
# Dokud podmínka platí, provádět příkazy.
$x = 0;
while($x < 5)
{
  print $x,"\n";
  $x++;
}
```

Příklad: cyklus `for`

- Provést první výraz (jednorázově na začátku). Zkontrolovat, zda platí podmínka ve druhém výrazu. Pokud ano, provést příkazy v těle cyklu. Po nich provést třetí výraz a skočit zpět na test podmínky.

```
for ($i=0; $i<=$#x; $i++)
{ print $x[$i], "\n"; }
foreach $x (@pole)
{ print "$x\n" }
```

Příklad: podmínka

```
if ($x == 1)
  { print "jedna\n"; }
elsif ($x == 2)
  { print "dva\n"; }
else
  { print "mnoho\n"; }
```

Relační operátory

- Porovnávání čísel jako v céčku: rovno **==**, nerovno **!=**, menší **<**, větší **>**, menší nebo rovno **<=**, větší nebo rovno **>=**.

```
"001"=="1"
```

- Porovnávání řetězců: rovno **eq**, nerovno **ne**, menší **lt**, větší **gt**, menší nebo rovno **le**, větší nebo rovno **ge**.

```
"001"ne"1"
```

Asociativní pole (hash)

- Hašovací tabulka.
- Seznam dvojic klíč – hodnota.
- Jméno proměnné začíná na `%`, ale hodnoty jsou skaláry, takže opět začínají `$`.
- Do pole x vložit slovo „dog“ a asociovat ho se slovem „pes“:

```
$x{"pes"} = "dog";
```

Příklad: asociativní pole

```
for($i=0; $i<=$#vstup; $i++) {
  @radek = split(",", $vstup[$i]);
  $cs = $radek[0];
  $en = $radek[1];
  $slovnik{$cs} = $en;
}
while(($cs,$en) = each %slovnik) {
  print $cs, " ", $en, "\n"; }
```

Na konci vrátí FALSE, pak znova od začátku.

Nesetříděné!

Třídění a výpis hashe

```
@klice = keys(%slovnik);
@x = sort
{
   return $slovnik{$a}
      cmp $slovnik{$b}; # <=>
   #  -1: $a lt $b; 0: $a eq $b; +1: $a gt $b
}
(@klice);
```

Čtení standardního vstupu

- Právě čtený řádek je ve zvláštní proměnné $_:

```
while(<STDIN>)
{
  print "> ", $_;
}
```

- Jméno souboru jako argument, není-li, pak STDIN:

```
while(<>)
{
  print;
}
```

Typické použití Perlu: filtr na textové soubory

```
preproc | program.pl | postproc
preproc | program.pl (na obrazovku)
cat vstup | program.pl | postproc
program.pl < vstup | postproc
program.pl vstup > vystup
```

`cat` (Unix) = `type` (Windows)

Čtení z pojmenovanéh

Ale konstantní cesty jsou zřídkakdy ospravedlnitelné!

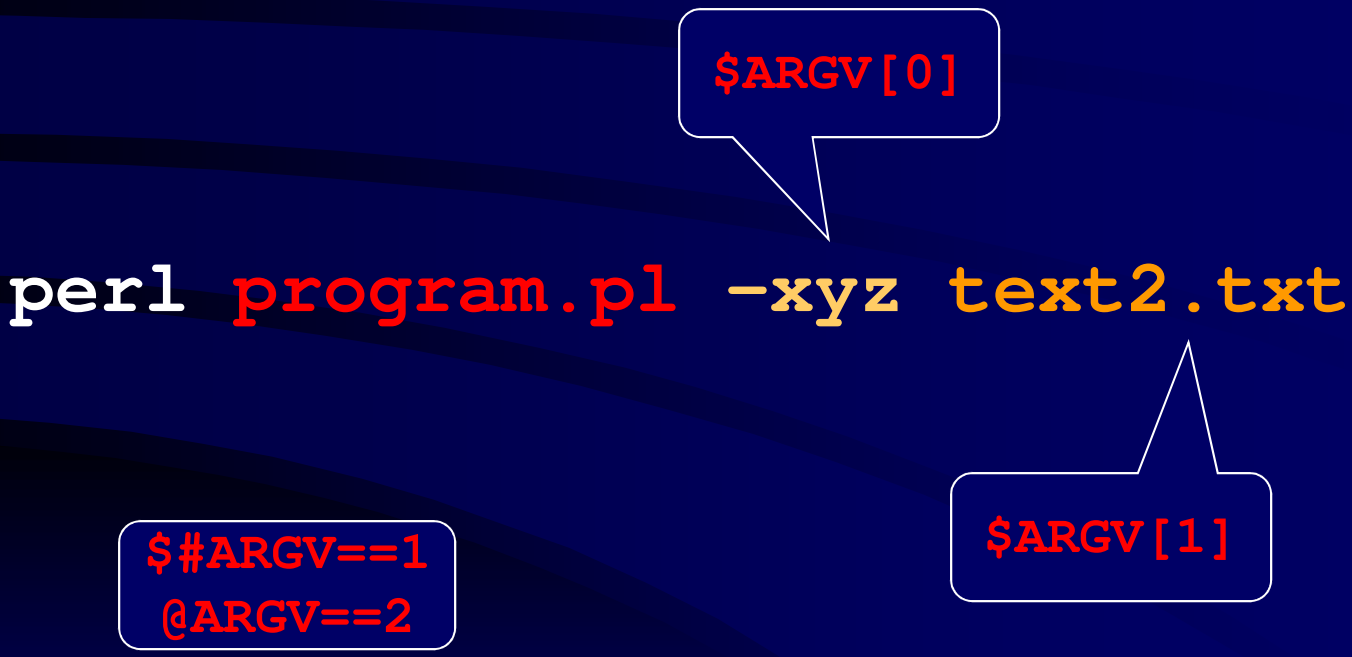
```
open(SOUBOR, "text.txt")
  or die("Nelze otevřít: $!\n");
while(<SOUBOR>)
{
   ...
}
close(SOUBOR);
```

Po dočtení souboru vrací už pořád FALSE, až než soubor znova otevřeme.

Převzetí parametrů z příkazového řádku

```
open(SOUBOR1, $ARGV[0]);
while(<SOUBOR1>) { $n1++; }
close(SOUBOR1);
open(SOUBOR2, $ARGV[1]);
while(<SOUBOR2>) { $n2++; }
close(SOUBOR2);
if($n1>$n2) {print "první delší";}
else        {print "druhý delší";}
```

Předání parametrů na příkazovém řádku



Volby a jejich argumenty

- Můžeme ručně procházet @ARGV a hledat argumenty začínající pomlčkou (volby)
  - Pracné: typicky chceme volby v libovolném pořadí, některé volby mají své argumenty, alternativní psaní atd.
  - Řešení: knihovna Getopt
    - V @ARGV nechá jen nepoužité argumenty
    - Umí boolovské, numerické i řetězcové volby

```
use Getopt::Long;
my $volba = 'DEFAULT';
GetOptions('lang=s' => \$volba);
```

Čtení výstupu z jiných programů

- Používá se mechanismus „roury“ — příklad: **`dir | more`**

```
open(SOUBOR, "dir |");
while(<SOUBOR>)
{
 print; # Totéž co print($_);
}
```

Regulární výrazy

`\s`	mezera nebo tabulátor
`^`	začátek řetězce
`$`	konec řetězce
`a`	písmeno *a*
`a?`	0 nebo 1 výskyt *a*
`a+`	1 nebo více *a* za sebou
`a*`	0 nebo více *a* za sebou
`(ab)+`	1 nebo více řetězců "ab"
`[^abc]`	jiný znak než *a*, *b* nebo *c*
`[a-z]`	libovolné malé písmeno
`.`	libovolný znak

Standardní porovnávání je hladové

Lze-li podmínky výrazu splnit více způsoby:

`a?`	najde-li *a*, vezme ho
`a+`	vezme tolik *a*, kolik je možné
`a*`	vezme tolik *a*, kolik je možné
`a??`	nemusí-li, nevezme ho
`a+?`	vezme jen tolik *a*, kolik je nutné
`a*?`	vezme jen tolik *a*, kolik je nutné

Regulární výrazy v podmínkách

- Zjistit, jestli řetězec $x obsahuje podřetězec "abc":

```
if ($x =~ /abc/) { … }
```

- Zjistit, jestli řetězec $x začíná podřetězcem "abc":

```
if ($x =~ /^abc/) { … }
```

- Zjistit, jestli řetězec $x začíná velkým písmenem:

```
if ($x =~ /^[A-Z]/) { … }
```

- Zjistit, jestli řetězec $x nezačíná malým písmenem (první ^ reprezentuje začátek řetězce, druhé negaci):

```
if ($x =~ /^[^a-z]/) { … }
```

Záměna řetězců pomocí regulárních výrazů

- Pokud je způsob "g", výsledkem operace je řetězec, v němž jsou *všechny* výskyty podřetězce *zdroj* nahrazeny podřetězcem *cíl*. Pokud je způsob prázdný, nahradí se pouze první výskyt.

```
s/zdroj/cíl/způsob
```

- Zaměnit v řetězci $x všechna a na b:

```
$x =~ s/a/b/g;
```

- Nahradit první a v řetězci $x písmenem b:

```
$x =~ s/a/b/;
```

Příklad: záměna pomocí regulárních výrazů

- Nahradit všechny řetězce po sobě jdoucích áček jediným písmenem a:

```
$x =~ s/a+/a/g;
```

- Odstranit všechny řetězce áček:

```
$x =~ s/a+//g;
```

- Odstranit mezery na začátku řetězce:

```
$x =~ s/^\s+//;
```

- N-tý uzávorkovaný úsek zdroje lze zkopírovat do cíle:

```
$x =~ s/<A>(.*?)<\/A>/<B>$1<\/B>/g;
```

Příklad 1: Četnosti slov

```
while(<STDIN>) {
    $_ =~ s/^\s+//; # Odstranit mezery na začátku.
    Pokud řádek začíná mezerami, první prvek pole slov by
    byl prázdný.
    @slova_na_radku = split(/\s+/,$_); #
    Rozdělit řádek na pole slov. Zatím neuvažujeme
    interpunkci, takže slovo jde od mezery do mezery!
    foreach $slovo (@slova_na_radku)
    { $cetnosti{$slovo}++; }
}
```

Příklad 2: Četnosti dvojic slov

```
while(<STDIN>) {
   $_ =~ s/^\s+//;
   @slova = split(/\s+/,$_);
   for ($i=0; $i<=$#slova-1; $i++) {
      $dvojice = $slova[$i]. " "
   .$slova[$i+1];
      $cetnosti{$dvojice}++;
   }
}
```

Příklad 3: Četnosti třípísmenných koncovek

```
while(<STDIN>) {
  $_ =~ s/^\s+//;
  @slova = split(/\s+/,$_);
  for($i = 0; $i <= $#slova; $i++) {
     $l = length($slova[$i]);
     $klic = $l<3 ? $slova[$i] :
           substr($slova[$i], $l-3, 3);
     $konc{$klic}++;
} }
```

Příklad 4: Lepší dělení na slova

```
while(<STDIN>) {
   s/([,;:\-.?!])/ $1/g; # operuje nad $_
   s/^\s+//;
   @slova = split(/\s+/, $_);
   foreach my $slovo (@slova) {
       print "<w>$slovo</w>\n";
   }
}
```

Cvičení

- Rozšiřte program na označování slov tak, aby všem celým číslům přiřadil značku **<n>**. Je-li číslo rozděleno mezerami (např. 1 650 000), odstraňte tyto mezery a číslo slepte.
- Před značku interpunkce vložte samostatný řádek se značkou **<D>** všude tam, kde byla původně interpunkce přilepena ke slovu. Tam, kde byla mezera před interpunkcí už v originále, nic nepřidávejte!