PB173 Perl

07 Operácie na reťazcoch, regulárne výrazy

Roman Lacko <xlacko1@fi.muni.cz>

Obsah

Reťazcové operácie

Reťazce v Perli sme doteraz používali pomerne intuitívne.
Pre zhrnutie:

- Reťazce sú skaláry, nie polia.
- Veľkosť je uložená interne, **nepoužívajú** terminálny bajt `\0`

```
say length "abc\0def";                  #  7
```

```
printf("%d\n", strlen("abc\0def"));     // 3
```

- Môžu sa chovať rôznym spôsobom podľa kódovania.

Reťazcové literály

Zatiaľ sme videli konštrukciu reťazcov pomocou úvodzoviek (*quotes*) a zodpovedajúcich operátorov (*quote operators*):

```
say 'User $ENV{USER}';          # User $ENV{USER}
say q"Use $ENV{USER}";          # Same thing

say "User $ENV{USER}";          # User pazuzu
say qq'Use $ENV{USER}';         # Same thing
```

Operátory `q` a `qq` nemenia svoj význam, ani keď sa skombinujú s `""` alebo `''`. Viď `ex01t-literals.pl`.

Špeciálne znaky v reťazcoch

V reťazcoch s interpoláciou sa môžu nachádzať rôzne špeciálne značky:

`\t`, `\n`, `\r`, ...
Tabulátor, koniec riadka, *carriage retturn*, ... (ako v C)

`\xNN`, `\NNN`
Bajt zadaný hexadecimálne alebo oktálne.

`\x{XXXX}`, `\o{OOOO}`
Unicode znak zadaný v hexadecimálnej alebo oktálnej sústave.

`\N{NAME}`, `\N{U+XXXX}`
Unicode znak zadaný názvom alebo jeho `U+NNNN` sekvenciou.

`\x` a `\o` nemusia byť vždy ASCII, záleží od platformy.
`\N{}` je vždy Unicode.

Here documents

Dlhšie odstavce textu je možné vypísať pomocou *Here documents* ako v Shelli:

```
print <<EOF;            # Or ‹<<"EOF"›, ‹<< "EOF"›.
This is a string literal spanning multiple lines.
This version supports $variable interpolation.
EOF

print <<\EOF;           # Or ‹<<'EOF'›, ‹<< 'EOF'›.
No interpolation here.
EOF
```

Okrem `EOF` môžeme použiť akýkoľvek iný reťazec.

Here documents

Od Perl 5.26 existuje konštrukcia aj pre odsadený *here document*:

```
if (CONDITION) {
    say <<~EOF;         # Or ‹<<~"EOF"›, ‹<<~ "EOF"›.
    This is an indented here document.
    EOF
}
```

Všetky riadky musia začínať rovnakými bielymi znakmi, aké sú pred `EOF`, okrem prázdnych riadkov.

Here documents

Môžeme mať viac než jeden here document, ako v ZSH:

```
print <<DOC1, <<DOC2;
The first document.
DOC1
The second document.
DOC2
```

Kódovanie a reťazce

```
length [EXPR]
```

- Vráti počet *znakov* v reťazci.

```
say length "αβγ";                                   # 6
say length "\316\261\316\262\316\263";              # Same thing

use utf8;
say length "αβγ";                                   # 3
```

Prevod medzi znakom a kódovou hodnotou

```
chr [NUMBER]
```

- Vráti znak zodpovedajúci zadanej hodnote.

```
ord [EXPR]
```

- Vráti kódovú hodnotu prvého *znaku* v zadanom výraze.
- Ak sa výraz vyhodnotí na prázdny reťazec, vráti 0.

```
use utf8;
my @code_points = map { ord } split '', "λουλούδι";
my @code_points = unpack "U*", "λουλούδι";
```

Číselné konverzie

Perl nerozlišuje medzi reťazcom a číslom,
tieto operácie menia *sémantiku* číselných hodnôt v reťazcoch.

```
hex [EXPR]
```

- Interpretuje reťazec ako číslo v hexadecimálnej sústave.
- Vráti nový skalár s danou hodnotou.
- Môže začať prefixom `0x`, `x`.

```
say hex "1f"            #  31
say hex "xab"           # 171
say hex "0xff"          # 255
```

Číselné konverzie

```
oct [EXPR]
```

- Interpretuje reťazec ako číslo v osmičkovej sústave.
- Čísla môžu začínať prefixom `0o` alebo `o`.
- Napriek názvu akceptuje aj `0x`, `x` pre hexadecimálnu a `0b`, `b` pre binárnu sústavu.

```
say oct "644"           # 420
say oct "b1001"         #   9
say oct "0xf0"          # 240
```

Odbočka: `int`

Pre úplnosť, existuje aj operátor `int`:

```
int [EXPR]
```

- Vráti celočíselnú časť výrazu zadaného ako EXPR.
- **Nepoužívajte** na zaokrúhľovanie!

Veľkosti písmen

```
lc [EXPR]
uc [EXPR]
```

- Vráti reťazec, kde sa všetky písmená zmenia na miniskuly (***l****ower* ***c****ase*) resp. majuskuly (***u****pper* ***c****ase*).

```
lcfirst [EXPR]
ucfirst [EXPR]
```

- Ako `lc` a `uc`, ale zmení len prvé písmeno.

Skúste `ex05-case-eq.pl`.

Fold case

```
fc [EXPR]
```

- Prevedie reťazec do jednoznačnej podoby určenej pre porovnávanie, bez ohľadu na veľkosť, tzv. *fold case*.

```
uc "ς"          # Σ
lc "Σ"          # σ, not ς!

fc "ς"          # σ
fc "Σ"          # σ
```

Ďalšie operácie

```
substr EXPR, OFFSET, [LENGTH, [REPLACEMENT]]
```

- Pre opakovanie: Extrahuje časť reťazca od `OFFSET` dĺžky `LENGTH`.
- Pracuje so *znakmi*, nie bajtami.
- Ak je zadaný štvrtý parameter, uvedenú časť reťazca nahradí (ako `splice`).

Ak `EXPR` je *l-value*, potom aj výsledok je *l-value*:

```
my $string = "Hi, World!";

substr $string, 0, 2, "Hello";          # Hello, World!
substr($string, 0, 2) = "Hello";        # Same thing
```

Ďalšie operácie

```
index STR, SUBSTR, [POSITION]
rindex STR, SUBSTR, [POSITION]
```

- Vrátia pozíciu SUBSTR v reťazci STR od začiatku resp. konca, prípadne od uvedenej pozície.

Ďalšie operácie

```
reverse [EXPR]
```

- Pre opakovanie: V zoznamovom kontexte obráti pole.
- V skalárnom kontexte spojí argumenty do reťazca a obráti reťazec.

```
sprintf FORMAT, LIST
```

- Ako `printf`, ale výsledný reťazec vráti.

Regulárne výrazy

Uhádnete, čo robí nasledujúca funkcia?

```
sub obscuro($n) {
    (1 x $n) !~ /^(?:1?|(11+)\1+)$/;
}
```

Formálne regulárne výrazy

- Stručne popisujú formálny regulárny jazyk.

```
c ∈ Σ, ε, ø     # bases of regular expressions
α, β → αβ       # α followed by β
     | α + β    # α or β
     | α*       # α repeated zero or more times
     | (α)      # explicit precedence
```

- Je možné ich algoritmicky previesť na konečný automat (FSA).

Regulárne výrazy v programovacích jazykoch

V programovacích jazykoch sú regulárne výrazy obvykle v knižnici.
Typicky majú rozšírenia nad rámec formálnych regulárnych výrazov.

V Perli sú regulárne výrazy priamo v syntaxi jazyka ako operátory,
tzv. *(regexp) quote-like operators*.

```
EXPR =~ PATTERN
EXPR !~ PATTERN          # ≈ !(EXPR =~ PATTERN)
```

- Aplikuje výraz PATTERN na EXPR.
- Môže sa aplikovať aj na $_:

```
SOMETHING if /REGEX/;
SOMETHING if !/REGEX/;
```

Syntax RE v Perli

Znaky v regulárnom výraze obvykle reprezentujú samy seba.
Výnimkou sú *metaznaky* (*metacharacters*), ktoré majú špeciálny význam:

`^`, `$`	Začiatok a koniec riadka
`.`	Ľubovoľný znak
`α|β`	Alternatíva
`[…]`, `[…-…]`, `[^…]`	Triedy znakov
`(…)`, `(?…)`	Skupiny
`α*`, `α?`, `α+`, `α{a,b}`	Opakovacie operátory
`\χ`	*Backslash sequences*

Význam metaznakov je možné vypnúť `\`, napr. `\|`.

Perl RE: ^, $, alternatíva

Začiatok a koniec riadka sú tzv. *zero-width assertions*:

```
"abc" =~ /b/;                   # True, match on a‹b›c.
"abc" =~ /^b/;                  # False
"abc" =~ /c$/;                  # True, match on ab‹c›.
```

Alternatíva vyberá medzi dvoma možnosťami, vždy najprv zľava:

```
"foot" =~ /o(a|o)/;             # True, match on f‹oo›t
"abc" =~ /^ab|dc$/;             # True, match on ‹ab›c
"abc" =~ /^a(b|d)c$/;           # True, match on ‹abc›
"aaaa" =~ /(a|aa|aaa)/;         # True, match on ‹a›aaa
```

Zátvorky zároveň vytvárajú *skupiny*.

Perl RE: Triedy znakov

Triedy znakov:

```
"perl" =~ /[pe]rl/;             # False
"hal3000" =~ /[a-z][0-9]/;      # True, match on ha‹l3›000
```

Zhodu na znaku `-` musíme tento znak uviesť na okrajoch alebo ako `\-`:

```
"mu-th-ur" =~ /[0-9]u/;         # False
"mu-th-ur" =~ /[09-]u/;         # True, match on mu-th‹-u›r
"mu-th-ur" =~ /[0\-9]u/;        # True, match on mu-th‹-u›r
```

Je možné k triedam vytvoriť aj doplnok:

```
"axbxcx" =~ /[^ab]x/;           # True, match on axbx‹cx›
"a^2" =~ /[^^]/;                # True, match on ‹a›^2
```

Perl RE: Triedy znakov

POSIXové triedy znakov:

```
"0xf" =~ /0x[[:xdigit:]]/;      # True, match on ‹0xf›
"end\0" =~ /[^[:print:]]/;      # True, match on end‹\0›
```

Perl navyše definuje tzv. *backslash sequences* s podobným významom:

```
"script.pl" =~ /\w\W/;          # True, match on ‹t.›
```

Tieto sekvencie môžu mať aj nulovú dĺžku v reťazci:

```
"Harry Potter" =~ /\w\b/;       # True, match on Harr‹y› Potter
"Hogwarts" =~ /\b/;             # True, match on ‹›Hogwarts
```

Perl RE: *Backslash sequences*

`\w`, `\d`
Podobné ako `[A-Za-z0-9_]` resp. `[0-9]`

`\s`
Akceptuje biele znaky okrem `\n`

`\b`
Akceptuje okraj slova

`\W`, `\D`, `\S`, `\B`
Opak `\w`, `\d`, `\s`, `\b`

`\A`, `\Z`
Akceptujú začiatok a koniec reťazca

`\G`
Akceptuje pozíciu, kde skončil posledný test zhody

Perl RE: Akýkoľvek znak

Metaznak `.` akceptuje akýkoľvek znak:

```
"abc" =~ /a.c/;                 # True, match on ‹abc›
```

Všeobecne však neakceptuje `\n`, kým sa nezapne príznak (neskôr).

Perl RE: Opakovacie kvantifikátory

α* opakuje vzor neobmedzený počet krát:

```
"aaab" =~ /a*b/;                    # True, match on ‹aaab›
"ac" =~ /b*c/;                      # True, match on a‹c›
```

α+ je skratka za αα*:

```
"ab" =~ /a+b/;                      # True, match on ‹ab›
"ac" =~ /b+c/;                      # False
```

α? je α alebo nič:

```
"abd" =~ /ab?c?d/;                  # True, match on ‹abd›
"abbc" =~ /ab?c/;                   # False
```

Perl RE: Rozsah

Rozsah pre opakovanie `{n}` a `{min,max}`:

```
"aaab" =~ /a{2}b/;              # True, match on a‹aab›
"aab" =~ /a{1,4}b/;             # True, match on ‹aab›
```

Práve jedno z `min` alebo `max` je možné vynechať, implicitne 0 resp. ∞.

- `α*` ≈ `α{0,}`
- `α+` ≈ `α{1,}`
- `α?` ≈ `α{,1}`

Zhoda so vzorom

```
EXPR =~ /REGEX/FLAGS
EXPR =~ m'REGEX'FLAGS   # Or m"regex", m{regex}, …
```

- V skalárnom kontexte vráti *true* práve ak EXPR vyhovuje výrazu REGEX.
- V zoznamovom kontexte sa chová zložitejšie.

```
if ($filename =~ /\.(pl|pm)$/) {
    say "Found Perl source '$filename'";
}
```

```
foreach (@lines) {
    process($_) unless !m'^(#|//)';
}
```

Capture groups

Skupiny v regulárnych výrazoch môžu meniť prioritu operácií v RE.
Zároveň však udržujú hodnoty zachytených výrazov.

Ak reťazec vyhovuje výrazu, potom pre každú z n zátvoriek vo výraze vznikne premenná `$τ` ($1 \leq \tau \leq n$):

```
say "Display width is $1, height is $2"
    if "display: 800x600" =~ /(\d+)x(\d+)/;
```

Pomocou `\g{τ}` (tzv. *backreference*) môžeme používať časti priamo vo výraze:

```
say "Found doubled character '$1'"
    if "foot" =~ /(.)\g{1}/;           # Or just ‹\g1›.
```

Perl dovoľuje aj `\1`, ale niekedy to môže byť nejednoznačné.

Capture groups **(rozšírenia)**

Skupiny v Perli majú rozšírenia, ktoré sa zapínajú `?` za zátvorkou:

`(?#γ)`	komentár
`(?:α)`	nezachytáva hodnotu do premennej
`(?<NAME>α)`	zachytí hodnotu do `\g{NAME}` a `$+{NAME}`
`(?'NAME'α)`	ako `(?<NAME>α)`
`(?|α|β|…)`	rovnaké číslovanie skupín v α, β atď.
`(?{ CODE })`	volanie kódu z RE
`(??{ CODE })`	dosadenie výrazu pri vyhodnocovaní RE
`(?(COND)α)`	podmienený výraz
`(?(COND)α|ω)`	podmienený výraz s `else` vetvou

Test okolia

Test, či sa v okolí vzoru nachádza iný vzor.

`(?=α)`	pohľad dopredu
`(?!α)`	negatívny pohľad dopredu
`(?<=α)`	pohľad dozadu
`(?<!α)`	negatívny pohľad dozadu

```
if ("giraffe" =~ /(.)(?=(.)\g2)/) {
    say "Letter followed by doubled letter: $1";        # a
    say "Entire match: $&";                             # a
}
```