PB173 Perl

07 Reťazce

Roman Lacko xlacko1@fi.muni.cz

2023-03-27

Obsah

Reťazcové operácie

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
```

- Operácie sa môžu správať rôzne podľa kódovania.

Reťazcové operácie

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`.

Reťazcové operácie: Špeciálne znaky

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.

Reťazcové operácie: 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›
No $variable interpolation here.
EOF
```

Okrem ‹EOF› môžeme použiť akýkoľvek iný reťazec.

Reťazcové operácie: Here documents

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

```
sub print_help() {
    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.

Reťazcové operácie: Here documents

Môžeme mať viac než jeden *tu dokument*, ako v ZSH:

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

Reťazcové operácie: ‹length›

```
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
```

Reťazcové operácie: Prevod

```
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";
```

Reťazcové operácie: Číselné konverzie

```
oct [EXPR]
hex [EXPR]
```

Interpretujú výraz ako reťazec v zadanej sústave.
Prefixy `0`, `0o`, `o` (`oct`) a `0x`, `x` (`hex`).

> **!** `oct` napriek názvu konvertuje aj hexadecimálne a binárne reťazce (prefix `0b` alebo `b`).

Reťazcové operácie: Číselné konverzie

Pre úplnosť, existuje aj operátor `int`:

```
int [EXPR]
```

Vráti celočíselnú časť výrazu zadaného ako `EXPR`.

Zaokrúhľovanie
Modul `POSIX` implementuje `floor()`, `ceil()` a `round()`.
Nepoužívajte `int`.

Reťazcové operácie: Veľkosti písmen

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

Zmenia všetky písmená na miniskuly (***l**ower **c**ase*) resp. majuskuly (***u**pper **c**ase*).

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

Ako ‹`lc`› a ‹`uc`›, ale len pre prvé písmeno

Reťazcové operácie: Fold case

```
fc [EXPR]
```

Prevedie reťazec do podoby určenej na porovnávanie, bez ohľadu na veľkosť písmen.

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

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

Reťazcové operácie: ‹substr›

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

Extrahuje časť reťazca od ‹OFFSET› dĺžky ‹LENGTH›.
Pracuje so *znakmi*, nie bajtami.

Ak je zadaný štvrtý parameter, uvedenú časť reťazca nahradí (ako ‹splice› pre pole).

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
```

Reťazcové operácie: ‹index›

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

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

Reťazcové operácie: ‹reverse›, ‹sprintf›

```
reverse [EXPR]
```

V skalárnom kontexte spojí argumenty do reťazca a obráti reťazec.
V zoznamovom kontexte obráti pole.

```
sprintf FORMAT, LIST
```

Ako ‹printf()›, ale výsledný reťazec vráti.

Regulárne výrazy

Regulárne výrazy

Uhádnete, čo robí nasledujúca funkcia?

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

A day may come when the courage of men fails, when we forsake our friends and break all bonds of fellowship, but it is not this day.
— Aragorn J. R. R. Tolkien: The Return of the King

Regulárne výrazy

Teoreticky regulárne výrazy (RE) stručne popisujú formálny regulárny jazyk.

- Pre neprázdnu konečnú abecedu Σ, výrazy c (c in
- Sigma]), ϵ a $\emptyset$ sú RE.
- Ak α, β sú RE, potom aj $(\alpha\beta)$, $(\alpha + \beta)$ a $(\alpha^\star)$ sú RE.

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

Prakticky sa často používajú s mnohými nekonzervatívnymi rozšíreniami.
V programovacích jazykoch typicky ako nejaká knižnica.

Regulárne výrazy: Perl

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

```
m{REGEX}FLAGS                    # m/…/ or /…/
s{REGEX}{REPLACEMENT}FLAGS       # s/…/…/
tr{SEARCH}{REPLACEMENT}FLAGS     # tr/…/…/ or y/…/…/
```

Regulárne výrazy: Bind operátor

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

Aplikuje operáciu na pravej strane na výraz na ľavej strane.
Presná sémantika závisí od pravej strany.

```
if ($value =~ /REGEX/) {    # m/REGEX/, m{REGEX}, …
    say "$value matches!";
}
```

Skrátené tvary sú užitočné pre ‹grep› alebo postfixové podmienky:

```
grep { /REGEX/ } LIST      #   $_ =~ /REGEX/
STATEMENT if !/REGEX/      # !($_ =~ /REGEX/)
```

Regulárne výrazy: Syntax

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
`\χ`	*Backslash sequences*
`(…)`, `(?…)`	Skupiny
`α*`, `α?`, `α+`, `α{a,b}`	Opakovacie operátory

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

Regulárne výrazy: ‹^›, ‹$›

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

```
"abc" =~ /b/;                    # Match a‹b›c
"abc" =~ /^b/;                   # No match
"abc" =~ /c$/;                   # Match ab‹c›
```

Regulárne výrazy: Alternatíva

Alternatíva vyberá medzi možnosťami, vždy najprv zľava:

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

Zátvorky zároveň vytvárajú *skupiny*.

Regulárne výrazy: Triedy znakov

Trieda znakov je množina možností:

```
"perl" =~ /[pe]rl/;             # No match
"hal3000" =~ /[a-z][0-9]/;      # Match ha‹l3›000
```

Špeciálny význam ‹-› zrušíme na okrajoch triedy alebo ako ‹\-›:

```
"mu-th-ur" =~ /[0-9]u/;         # No match
"mu-th-ur" =~ /[09-]u/;         # Match mu-th‹-u›r
"mu-th-ur" =~ /[0\-9]u/;        # Match mu-th‹-u›r
```

Doplnok tried sa vytvorí ‹^› na začiatku:

```
"axbxcx" =~ /[^ab]x/;           # Match axbx‹cx›
"a^2" =~ /[^^]/;                # Match ‹a›^2
```

Regulárne výrazy: Triedy znakov

Niektoré triedy znakov majú v POSIX meno:

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

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

```
"script.pl" =~ /\w\W/;          # Match ‹t.›
```

Tieto sekvencie môžu mať aj nulovú dĺžku v reťazci:

```
"Harry Potter" =~ /\w\b/;       # Match Harr‹y› Potter
"Hogwarts" =~ /\b/;             # Match ‹›Hogwarts
```

Regulárne výrazy: Triedy znakov

`\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
`\Q`, `\E`	Ohraničuje časť, kde metaznaky stratia svoj význam
`\G`	Akceptuje pozíciu, kde skončil posledný test zhody

Regulárne výrazy: ‹ . ›

Metaznak ‹ . › akceptuje akýkoľvek znak:

```
"abc" =~ /a.c/;                    # Match ‹abc›
```

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

Regulárne výrazy: Opakovacie kvantifikátory

‹α*› opakuje vzor neobmedzený počet krát:

```
"aaab" =~ /a*b/;                # Match ‹aaab›
"ac" =~ /b*c/;                  # Match a‹c›
```

‹α+› je skratka za ‹αα*›:

```
"ab" =~ /a+b/;                  # Match ‹ab›
"ac" =~ /b+c/;                  # No match
```

‹α?› je ‹α› alebo nič:

```
"abd" =~ /ab?c?d/;              # Match ‹abd›
"abbc" =~ /ab?c/;               # No match
```

Regulárne výrazy: Rozsah

Rozsah pre opakovanie ‹α{n}› a ‹α{min,max}›:

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

Práve jedno z ‹min› alebo ‹max› je možné vynechať,
implicitne sa použije 0 resp. ∞.

‹α*› $\approx$ ‹α{0,}›
‹α+› $\approx$ ‹α{1,}›
‹α?› $\approx$ ‹α{,1}›

Regulárne výrazy: Pažravé a lenivé vzory

Výrazy `α*`, `α+`, `α?` a `α{N,M}` sú takzvané „pažravé“ (*greedy*).
Pokúsia sa najprv akceptovať čo najdlhší podreťazec.

```
say $& if "aaab" =~ /a*ab/;
#           ^^^       a*
#              ×        ab   (backtracking)
#           ^^        a*
#             ^^        ab   (match)
```

Backtracking
Trieda algoritmov, ktoré prehľadávajú (diskrétny) stavový priestor.
Z neperspektívnej cesty sa vracajú a skúšajú iné.
Perl RE používa backtracking pri výbere prechodov v automate

Regulárne výrazy: Pažravé a lenivé vzory

Pridaním ‹?› (napr. ‹α+?›) vytvoríme naopak lenivý (*lazy*, *non-greedy*) vzor.

```
say $& if "aaab" =~ /a*?ab/;
#                    a*   (zero-width match)
#           x          ab (backtracking)
#           ^       a*
#            x        ab (backtracking)
#           ^^       a*
#             ^^       ab (match)
```

Regulárne výrazy: Capture groups

Zátvorky 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› ($1 \leq \tau \leq n$) s **dynamickým rozsahom**:

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

Regulárne výrazy: Capture groups

Pomocou ‹\g{N}› alebo ‹\gN› môžeme zachytené časti používať vo zvyšku výrazu (tzv. *backreference*):

```
if ("foot" =~ /(.)\g{1}1/)) {           # ‹\g1› would not work here!
    say "Found doubled character '$1' before 1";
}
```

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

Regulárne výrazy: Premenné

Perl **po vyhodnotení** `=~` s regulárnym výrazom nastavuje špeciálne premenné:

`$1`, `$2`, ... hodnoty zachytené skupinami

`$+`, `$^N` hodnota zachytená najvyššou resp. poslednou zatvorenou skupinou

```
"ab" =~ /^((.)(.))$/;   # $1  = 'ab', $2 = 'a', $3 = 'b'
                        # $^N = 'ab',           $+ = 'b'
```

`$&` podreťazec, ktorý vyhovel výrazu

`$``, `$'` prefix a postfix vyhovujúceho reťazca

```
say "$`$&$'" if /PATTERN/;  # Should be equal to $_
```

`@-`, `@+` Začiatočné resp. koncové pozície skupín (0 pre `$&`)

`%-`, `%+` Všetky resp. najľavejšia hodnota pomenovanej skupiny

Úvodzovkovité operátory

Úvodzovkovité operátory

`q{STRING}`	≈ `'STRING'`
`qq{STRING}`	≈ `"STRING"`
`qw{α β γ…}`	≈ `('α', 'β', 'γ', …)`
`qx{COMMAND…}`	≈ `` `COMMAND…` ``, `system("COMMAND…")`

`m{REGEX}FLAGS`	zhoda s výrazom
`s{REGEX}{SUB}FLAGS`	nahradenie
`tr{FROM}{TO}FLAGS`	transliterácia znakov
`qr{REGEX}`	regulárny výraz ako hodnota

Namiesto `{}` je možné používať aj iné párové značky a znaky:
`q[…]`, `qq{…}`, `qw/…/`, `s/…/…/`, `m!…!`, `tr%…%…%`

Zhoda

```
m{PATTERN}FLAGS
/PATTERN/FLAGS
```

V skalárnom kontexte vráti logickú hodnotu, či reťazec vyhovuje výrazu.
V zoznamovom kontexte vráti hodnoty zachytené skupinami `()`.

Podobne ako pri ostatných *quote operators*, aj tu je možné namiesto `m//`
použiť iné znaky bez efektu na správanie, s výnimkou `''`.

```
my $string = "b";
"abc" =~ m"a$string";          # Match ‹ab›c
"abc" =~ m'a$string';          # No match
```

Príznaky

‹FLAGS› za koncom úvodzovkovitých operátorov mení ich správanie.
Tieto príznaky je možné kombinovať.

‹g›	test na viacnásobnú zhodu
‹i›	zhoda bez dôrazu na veľkosť písmen
‹m›	chápe reťazec ako viac riadkov (mení správanie ‹^› a ‹$›)
‹s›	chápe reťazec ako jeden riadok (mení správanie ‹.›)
‹n›	vypne zachytávanie hodnôt v skupinách (okrem pomenovaných)
‹x›, ‹xx›	povolí komentáre a medzery vo výraze

Zhoda: Príznak ‹g›

Tento príznak vyhodnocuje vzor opakovane.

V skalárnom kontexte povoľuje viacnásobné vyhodnotenie,
vždy pokračuje od poslednej pozície

```
say "$&" while "abc" =~ /./;    # ‹a›, ‹a›, ‹a›, …
say "$&" while "abc" =~ /./g;   # ‹a›, ‹b›, ‹c›
```

V zoznamovom kontexte vráti zachytené hodnoty skupín.
Ak vzor žiadne neobsahuje, potom vráti všetky vyhovujúce podreťazce.

```
say for "abc" =~ /./g;          # ‹a›, ‹b›, ‹c›
say for "abcd" =~ /(.)./g;      # ‹a›, ‹c›
```

Zhoda: Príznaky ‹m› a ‹s›

Príznak ‹m›

‹^› a ‹$› znamenajú začiatok a koniec *riadka*, nie *reťazca*.

Príznak ‹s›

‹.› akceptuje aj ‹\n›.

Čo akceptuje ‹/^a.b/ms›?

Zhoda: Príznaky ‹x› a ‹xx›

Povolí medzery a komentáre začínajúce ‹#› v regulárnom výraze.
‹/xx› navyše povolí medzery aj v ‹[ ]›.

```
say "$&" if $string =~ m{
    ^               # Start of the string
    \w              # A word character
    [0-9 a-f A-F]   # A hexadecimal digit
    \Q#\E           # Literal ‹#›
}xx;
```

Nahradenie

```
s/PATTERN/REPLACEMENT/FLAGS
```

Ak reťazec vyhovuje `m{PATTERN}`, vyhodnotí `REPLACEMENT`
a výsledkom nahradí vyhovujúcu časť reťazca.

> **!** Nahradenie modifikuje reťazec na ľavej strane `=~`!

```
my $str = "this or that";
say $str if $str =~ s/or/and/;          # this and that
```

Vráti počet vykonaných nahradení (0 alebo 1, s `/g` prípadne viac).

Nahradenie

Výraz v ‹REPLACEMENT› môže obsahovať okrem reťazca aj interpolované premenné:

```
my $string = "armadillo";
say $str if $str =~ s/(.)\g1/$1²/;      # armadil²o
```

Môže tiež obsahovať špeciálne sekvencie **pred premennými**:

‹\l›, ‹\u›	zmenia veľkosť nasledujúceho znaku.
‹\L›, ‹\U›	menia veľkosť písmen až do ‹\E›.
‹\F›	zapne *fold-case* až po ‹\E›.
‹\Q›	vypne význam špeciálnych znakov až po ‹\E›.

```
say $str if $str =~ s/^(.)/\U$1\E/;     # ARMADIL²O
```

Nahradenie: Príznaky

Operátor ‹s///› podporuje všetky príznaky ako ‹m//›, a navyše:

‹r› vytvorí kópiu reťazca a tú po zmene vráti

```
say "Cat" =~ s/t/lf/;   # Error: Can't modify constant item …
say "Cat" =~ s/t/lf/r;  # Calf
```

‹e›, ‹ee› vyhodnotí ‹REPLACEMENT› ako Perl výraz (raz resp. dvakrát)

```
say "<e,4>" =~ s/<([^,]+),(\d+)>/$1 x $2/re;     # eeee

say '<$ENV{USER}>' =~ s/<(.+?)>/$1/re;           # $ENV{USER}
say '<$ENV{USER}>' =~ s/<(.+?)>/$1/ree;          # pazuzu
```

Špeciality

Regulárne výrazy: Extended capture groups

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

`(?#γ)`	komentár
`(?:α)`	vypne zachytávanie hodnoty
`(?<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|β)`	podmienený výraz s `else` vetvou

Regulárne výrazy: Test okolia

Test, či sa v okolí vzoru nachádza iný vzor.
Nezachytáva žiadnu hodnotu.

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

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

Regulárne výrazy: Riadiace slovesá

Control verbs umožňujú zmeniť správanie a vyhodnotenie výrazu.
Sú tvaru `(*VERB)` alebo `(*VERB:ARG)`.

`(*PRUNE)`	Ореže a upne rozhodovací strom (ako `!` v Prologu)
`(*SKIP)`	Po zlyhaní začne hľadať od tejto znaky
`(*MARK:NAME)`, `(*:NAME)`	Vytvorí značku
`(*THEN)`	Pri zlyhaní skočí na ďalšiu vetvu v `(?|α|β|γ|…)`
`(*COMMIT)`	Pri zlyhaní ukončí s neúspechom celé prehľadávanie
`(*ACCEPT)`	Okamžite akceptuje
`(*FAIL)`, `(*F)`	Okamžite odmietne (spustí backtracking).

Všetky majú `(*VERB:ARG)` variant, ktorý nastavuje `$REGMARK`, okrem `(*FAIL:ARG)`, ktoré nastavuje `$REGERROR`.

‹qr{}› operátor

Operátor ‹qr{}› vráti regulárny výraz výraz ako hodnotu,
ktorú je možné uložiť do skalárnej premennej a použiť neskôr namiesto ‹m//›.

```
my $regex = qr{a.*x};

if ($string =~ $regex) {
    # …
}

say ref $regex;          # Regexp
```

Dereferencia vráti normalizovaný reťazec, ktorý sa môže môže meniť
s verziou Perlu.

Transliterácia

```
tr/SEARCH/REPLACE/FLAGS
y/SEARCH/REPLACE/FLAGS
```

`tr//` a `y//` sú synonymá.

Nahradí znaky v `SEARCH` zodpovedajúcimi znakmi v `REPLACE`, alebo ich zmaže (podľa príznakov). Nepoužíva interpoláciu.

```
my $string = 'abba';
say $string =~ tr/a-c/0-2/r;          # 0110
say $string =~ tr/$ab/0-2/r;          # 1221
```

Transliterácia

Ak je zoznam ‹REPLACE› kratší než ‹SEARCH›, doplní sa posledným znakom alebo kópiu ‹SEARCH›:

```
$string =~ tr/a-e/01/;          # Same as ‹tr/abcde/01111/›
$string =~ tr/abc//;            # Same as ‹tr/abc/abc/›
```

Transliterácia: Príznaky

‹c› nahradí komplement ‹SEARCH›
‹d› nedoplní kratší ‹REPLACE›, ale znaky zmaže
‹r› upraví kópiu reťazca (ako pri ‹s///›)
‹s› stlačí násobný výskyt nahradených znakov na jeden

```
say "abc" =~ tr/ab/x/r;                 # xxc
say "abc" =~ tr/ab/x/dr;                # xc
say "abc" =~ tr/ab/x/cr;                # abx

say "mississippi" =~ tr/ips/ha/sr;      # mhahahah
say "mississippi" =~ tr[s][p]sr;        # mipipippi
```