PB173 Perl

02 Zoznamy, polia

Roman Lacko xlacko1@fi.muni.cz

2023-02-22

Obsah

Zoznamy

Zoznamy

Zoznam (*List*)

Zoznam je sekvencia **skalárnych** hodnôt v zátvorkách oddelená ‹ , ›:

```
()
(1, 2, 3)
("red", "green", "blue")
```

Hodnoty nemusia mať rovnaký „typ“:

```
(42, "Perl", $x, \$y, \%c, [1], {}, sub ($n) { ... })
```

Zoznam

Zoznamy sú „ploché“:

```
(1, 2, (3, 4), 5)
(1, 2, 3, 4, 5)                 # Same thing
```

Prázdne prvky sa ignorujú:

```
("dramatic",,,,,,,,, "pause")
("dramatic", "pause")           # Same thing
```

Zoznam: Indexácia

Zoznamy je možné indexovať operátorom `[]`:

```
say ((1, 2, 3)[0])  # 1
```

Index môže byť aj záporné číslo, vtedy sa počíta „od konca“:

```
say ((1, 2, 3)[-1]) # 3
```

Index mimo rozsah vráti `undef`.

Zoznam: Výrezy

Indexovať je možné aj iným zoznam
(pre viac než jeden index môžeme zátvorky vynechať):

```
my ($a, $b) = (1, 2, 3, 4, 5)[1, 2];    # (2, 3)
```

Výsledkom je nový zoznam, ktorý voláme **výrez** (*slice*).

Zoznam: Operátor opakovania

```
LIST x COUNT
```

Vytvorí nový zoznam opakovaním prvkov:

```
(1, 2) x 3      # (1, 2, 1, 2, 1, 2)
```


Ľavá strana nemusí byť len zoznam

Operátor môže mať na ľavej strane aj skalár:

- ‹`(LIST) x N`› vytvorí zoznam hodnôt
  ‹`("ha") x 3`› → ‹`("ha", "ha", "ha")`›
- ‹`SCALAR x N`› opakuje skalár ako reťazec
  ‹`"ha" x 3`› → ‹`"hahaha"`›

Zoznam: ‹qw› operátor

Quote Word (‹qw›) operátor

```
qw{WORD WORD…}
```

Vytvorí nový zoznam z uvedených slov, oddelených medzerou.

```
qw{Hello World}     # ("Hello", "World")
qw{3 9 6}           # (3, 9, 6)
```

Okrem ‹{}› je možné použiť aj niektoré iné ASCII znaky
‹qw()›, ‹qw[]›, ‹qw<>›, ‹qw""›, ‹qw//›, …

Odbočka: Úvodzovkovité operátory

`q{STRING}` Ekvivalent `'STRING'`, tj. neinterpoluje (ani `q""`)

`qq{STRING}` Ekvivalent `"STRING"`, tj. interpoluje (aj `qq''`)

`qw{WORDS…}` Vráti zoznam slov z `WORDS…` rozdelených podľa medzery

Perl zátvorky počíta

```
q{a {b} c}          # 'a {b} c'
```

Zoznamy: ‹ . . › operátor

Range operator (‹ . . ›)

```
(START .. END)
```

Vráti zoznam čísel od ‹START› po ‹END› **vrátane**.

```
say foreach (1 .. 5);       # (1, 2, 3, 4, 5)
```

- Funguje aj na reťazcoch (*magic auto-increment*)
- Funguje inak v *skalárnom kontexte* (*flip-flop operator*)
- Existuje ešte ‹ . . . › (variant *flip-flop*)

Polia

Polia

Pole je premenná, ktorá drží zoznam. Má *sigil* ‹`@`›:

```
my $scalar;
my @array;
```

Pole môžeme inicializovať iným zoznamom alebo poľom:

```
my @primes = (2, 3, 5, 7, 11);
my @langs = qw(Perl C C++ Java);
my @numbers = (1 .. 42);
my @copy = @numbers;
```

Pole: Súvis so zoznamom

Pole je **premenná**, ktorá drží **zoznam**.

Pole môžeme obvykle použiť tam, kde sa očakáva zoznam, napríklad:

- Výrezy

```
@array[1, 2, 3]
@array[@indices]
```

- Volania funkcií

```
function(1, 2, 3);
function(@params);
```

Pole: Interpolácia

Celé pole môžeme vypísať interpoláciou:

```
my @names = qw{Alice Bob Craig};
say "@names";       # Alice Bob Craig
```

Pri interpolácii sa medzi prvky vkladá hodnota špeciálnej premennej `$"`.

```
$" = ':';
say "@names";       # Alice:Bob:Craig
```

Bude výpis iný pre `say @names;`?

Pole: Indexácia

K prvkom poľa pristúpime operátorom ‹`[]`›.

Sigil nehovorí Perlu len typ premennej, ale aj typ *hodnoty*!

```
my @numbers = (1 .. 4);
say @number;             # Okay, we have an array.
#say @numbers[0];        # This is NOT okay, because…
say $numbers[0];         # …we actually get a «scalar»

say @numbers[0, 1];      # Okay, we get a «list» (a slice)
```

Ako vytvoríme výrez poľa veľkosti 1?

Posledný index poľa

```
my @numbers = ("tic", "tac", "toe);
say $#numbers;      # 2
```

Pre prázdne pole vráti -1

```
my @twilight_films_i_like;
say $#twilight_films_i_like;        # -1
```

Počet prvkov zoznamu
Pole v *skalárnom kontexte* (neskôr) sa vyhodnotí na počet prvkov.

Priradenie na neexistujúci index

Perl pole automaticky zväčšuje podľa potreby.

Ak priradený index spôsobí vznik nových buniek, bude ich iniciálna hodnota `undef`.

```
my @rings = qw{vilya nenya narya};
$rings[666] = "the-one";
```

Pole v príklade vyššie má 667 prvkov.
Len indexy 0, 1, 2 a 666 majú definované hodnoty.

Pole: Iterácia

Platí aj pre zoznamy

```
foreach my $value (@array) {
    # ...
}
```

‹$value› je *alias* hodnoty, zmena sa prejaví aj v poli.

Pre pripomenutie, postfixová iterácia:

```
STATEMENT for (@array);
```

Hodnota v implicitnej premennej ‹$_›.

Pole: (Ne)interakcia so skalárom

Premenné s rovnakým názvom, ale rôznymi *sigils* sú v Perli **rôzne premenné**:

```
my $a = 42;
my @a = qw{foo bar};
sub a() { "Aaaaaaaa"; }

say $a;      # 42
say $a[1];   # bar
say a();     # Aaaaaaaa
```

V čistom kóde nepoužívajte!

Type Globs

Meno premennej je „odkaz“ do tabuľky symbolov, kde má každý *sigil* vlastnú hodnotu. Záznam v tejto tabuľke sa volá *Type Glob*.

Pole: Viacrozmerné polia

Pole **nemôže** obsahovať iné pole:

```
my @a = (2, 3);
my @b = (1, @a, 4);      # Creates a new flat list

say "@b";                # 1 2 3 4
```

Pripomienka

1. Pole je premenná, ktorá drží **zoznam**
2. Zoznam je sekvencia **skalárnych** hodnôt

Viacrozmerné polia sa však dajú vyrobiť inak (nabudúce):

- Pole môže obsahovať referenciu na iné pole
- Hash s emuláciou pomocou `$;`

Pole: Rozdiely oproti zoznamu

Čistý zoznam je imutabilný (nedá sa modifikovať):

```
(0 .. 5)[0] = 3;      # Can't modify list slice …
```

Zoznam v poli sa zmeniť dá:

```
my @numbers = (0 .. 5);
$numbers[0] = 3;      # OK
```

Pole: Rozdiely oproti zoznamu

Ako sa pole a zoznam správajú, keď ich priradíme do **skalárnej** premennej?

```
my $list = (5, 6, 7, 8);
say $list;              # 8

my @numbers = (5, 6, 7, 8);
my $array = @numbers;
say $array;             # 4
```

Prečo sú ‹`$list`› a ‹`$array`› odlišné? Čo ich hodnota znamená?

Kontext

Perl rozlišuje **kontext**, v ktorom sa výraz nachádza:

prázdny (*void*) kontext
: Hodnota výrazu sa nepoužije ani neuloží.

skalárny kontext
: Hodnota výrazu sa použije ako skalár.

zoznamový kontext
: Hodnota výrazu sa použije ako zoznam.

Kontext: Zoznam

Zoznam v skalárnom (a prázdnom) kontexte vyhodnotí všetky prvky a vráti posledný:

```
my $x = (1, 2, 3);      # 3
```


Operátor `,` v skalárnom kontexte (mimo zoznamu) sa správa ako sekvenčný operátor:

```
my $x = 1, 2, 3;        # → (my $x = 1), 2, 3
```

Zoznam v zoznamovom kontexte *postupne* priradí svoje prvky na ľavú stranu:

```
my ($x, $y) = (1, 2, 3);    # $x = 1, $y = 2
my @array = (1, 2, 3);
```

Kontext: Pole

Pole v skalárnom kontexte vráti počet prvkov:

```
my $size = @array;
```

Pole v zoznamovom kontexte sa správa ako zoznam:

```
my @new_array = ('1', '2', @l1, @l2);
```

```
my @args = ("Hello, %s\n", "Hello", $name);
printf @args;
```

Kontext: Implicitne

Implicitný kontext sa nastaví podľa použitia výrazu

Skalárny kontext

- Priradenie do skalárnej premennej
- Operátory, ktoré očakávajú skalár

```
3 + @array          # Scalar context
if (@array == 3)    # Scalar context
```

Kontext: Implicitne

Implicitný kontext sa nastaví podľa použitia výrazu

Zoznamový kontext

- Priradenie do poľa
- Argumenty funkcie (bez prototypov)

```
sum @numbers        # List context
sum(@numbers)       # List context
```

Prototypy?
Niektoré hrôzy je lepšie nechať na neskôr

Kontext: Explicitne

Skalárny kontext je možné vynútiť operátorom ‹scalar›:

```
scalar EXPRESSION
```

Zoznamový kontext špeciálny operátor nemá,
obvykle stačí použiť zátvory (‹(EXPRESSION)›).

```
$foo            # A scalar
($foo)          # A list
```

‹perlsecret› → *Goatse Operator*

Pole: Operácie

Vkladanie a výber prvkov z konca poľa:

```
push ARRAY, LIST
pop ARRAY
```

Vkladanie a výber prvkov zo začiatku poľa:

```
unshift ARRAY, LIST
shift ARRAY
```

Pole: Operácie

Pole môže obsahovať akékoľvek skalárne hodnoty, vrátane `undef`.

```
defined $array[INDEX]
```

- Otestuje, že hodnota v poli je definovaná.

> ❗ Prístup mimo pole sa vždy vyhodnotí na `undef`.

```
exists $array[INDEX]
```

- Otestuje, že pole má hodnotu na indexe `INDEX`
  (aj keby bola `undef`)

Pole: Operácie

```
splice ARRAY, OFFSET, [LENGTH, [LIST]];
```

- Z poľa ‹`@array`› zmaže prvky od pozície ‹`$offset`›.
  Ak ‹`$offset < 0`›, tak sa berie ‹`-$offset`› od konca.
- Pre parameter ‹`$length`› zmaže len zadaný počet prvkov.
  Ak ‹`$length < 0`›, maže do konca okrem posledných ‹`-$length`› prvkov.
- Ak je zadaný aj zoznam, tak zmazané prvky sa nahradia prvkami zoznamu.
- Vráti všetky zmazané prvky alebo posledný podľa kontextu.

Pole: Operácie

```
reverse LIST
```

- V zoznamovom kontexte obráti zoznam
- V skalárnom kontexte spojí hodnoty do reťazca a vráti opačný reťazec

```
say reverse qw{Hello World};         # WorldHello
say scalar reverse qw{World Hello}; # dlroWolleH
```

Pole: Operácie

```
split STRING, EXPRESSION, [LIMIT]
```

- Rozdelí hodnotu výrazu ‹EXPR› podľa znaku alebo reťazca v ‹STRING›.
- Vráti zoznam všetkých hodnôt, alebo len prvých ‹LIMIT› z nich

‹split› sa častejšie používa s regulárnymi výrazmi.

```
join STRING, LIST
```

- Spojí hodnoty v zozname reťazcom ‹STRING› a vráti nový reťazec.

Pole: Argument funkcie

Pole sa môže nachádzať ako **posledný** argument funkcie tzv. *slurpy parameter*.

```
sub range_filter($min, $max, @values) { … }

range_filter 10, 30, 1 .. 50;
```

Ako by sme implementovali vlastnú funkciou ‹push›?

```
push @array, 1, 2, 3;
```

Špeciality

Špeciality: Historické okienko

Nastaviteľný počiatočný index
Zastaraná vlastnosť od ‹v5.12›, vypnutá od ‹v5.30›

```
my @array = (1 .. 5);
$[ = 3;

say $array[2];  # undef
say $array[3];  # 1
say $array[4];  # 2
```

Špeciality: Iterácia cez pole

Iterácia s viacerými prvkami naraz
Táto vlastnosť je experimentálna od `v5.36`

```
use experimental 'for_list';

foreach my ($v1, $v2) (1 .. 5) {
    say "($v1, $v2)";     # (1, 2) (3, 4) (5, undef)
}
```

Toto rozšírenie môžete používať v úlohách

Špeciality: Iterácia cez pole

Iterácia s indexom
Táto vlastnosť je experimentálna od ‹v5.36›

```
use experimental qw(for_list builtin);

foreach my ($index, $value) (builtin::indexed @array) {
    say "$index, $value";
}
```

Toto rozšírenie môžete používať v úlohách

Špeciality: Unicode úvodzovky

Unicode úvodzovky
Experimentálna vlastnosť od ‹v5.36›

```
use utf8;
use experimental 'extra_paired_delimiters';

qw‹gryffindor hufflepuff ravenclaw slytherin›
qw«καλημέρα καληνυχτα»
```

Špeciality: ‹wantarray›

‹wantarray› je špeciálny operátor, ktorý funkcii povie, v akom kontexte sa použila. Podľa toho môže vrátiť rôzne hodnoty.

‹undef›	‹void› context
nepravda	skalárny kontext
pravda	zoznamový kontext

```
sub context() { wantarray ? "list" : "scalar or void"; }

say context();          # list
say scalar context();   # scalar or void
```