PB173 Perl

01 Úvod

Roman Lacko xlacko1@fi.muni.cz

2023-02-17

Obsah

1. Úvod

2. Prečo Perl?

3. Základy, výrazy, príkazy

4. Funkcie

5. Špeciality

Úvod

Úvod

Python is executable pseudocode.
Perl is executable line noise.
— Bruce Eckel

Úvod: Ciele predmetu

- Naučiť sa nový jazyk!
- Vyvrátiť zlú povesť Perlu

Predpoklady

- IB111 Základy programování
- Znalosť iného skriptovacieho jazyka (Python, Ruby)
- Základy shellu
- Git

Úvod: Cvičenia

- 3 bloky × 4 semináre
- Slidy v študijných materiáloch IS MU
- Zadania cvičení v GitLab FI

Úvod: Materiály

- ‹exNNt-*› komentované ukážky
- ‹exNNp-*› príklady na cvičenie
- ‹exNNx-*› doplňujúce alebo pokročilé ukážky

‹hwNNa-*›,… základné domáce úlohy

‹hwNNx-*›,… pokročilé domáce úlohy

Úvod: Bodovanie

10 b jedna **základná** úloha z každého týždňa podľa vlastného výberu

5 b každá *ďalšia* vypracovaná základná úloha z toho istého týždňa

10 - 30 b každá **ťažká** domáca úloha, bodovanie bude upresnené v zadaní úlohy

2 b vypracovanie príkladu na projektor, aktivita na hodine atď.

Úvod: Domáce úlohy

Jadrom hodnotenia sú domáce úlohy:

- Koniec odovzdávania **dva týždne** po zverejnení zadania.
- Úlohy sa odovzdávajú ako *Merge Request* v repozitári.
- Odovzdanie môžete po prvej recenzii opraviť.

Úvod: Hodnotenie

Na úspešné hodnotenie je treba splniť nasledujúce podmienky:

1. Nanajvýš **dve** neospravedlnené neúčasti na seminári.
2. Získať aspoň **80 bodov**.
   Ekvivalent plných 8 z 12 základných úloh.

Náhrada neúčasti

Neospravedlnenú neúčasť si môžete nahradiť vypracovaním ťažkej úlohy z toho istého bloku za 0 bodov.

Ak budete túto možnosť chcieť využiť, dajte mi najprv vedieť.

Prečo Perl?

Prečo Perl?

- Prakticky zameraný jazyk
- Rôzne úrovne abstrakcie a štýlov
- Veľké množstvo rozširujúcich balíkov (CPAN)

Prečo Perl? Použitie

- Rýchle skriptovanie, prototypovanie
- Webové aplikácie (CGI skripty)
- Práca s DB (Perl DBI)
- „Lepidlový“ jazyk
- Spracovanie textov

IMDb, DuckDuckGo, Bugzilla, AoC, IS MU

Write-only paradigm?

Čo robí nasledujúci program?

```
#include <stdio.h>
#include <stdlib.h>
#define B 6945503773712347754LL
#define I 585983823119196245 9LL
int main(int b,char**i){
    long long n=B,a=I^n,r=(a/b&a)>>4,y=atoi(*++i),
        _=(((a^n/b)*(y>>0)|y>>7)&r)|(a^r);
    printf("%.8s\n",(char*)&_);
}
```

Write-only paradigm?

Čo robí nasledujúca funkcia?

```
from functools import reduce

def a(n):
    return reduce( \
        (lambda r,x: r-set(range(x**2,n,x)) if (x in r) else r), \
        range(2,int(n**0.5)), set(range(2,n)) \
    )
```

Write-only paradigm?

Napísať nepochopiteľný kód je možné v **každom** jazyku.
V Perli je to len trochu jednoduchšie.

Ciele v seminári:

- Vhodné či nevhodné prístupy v ukážkach kódov.
- Dôraz na čistotu kódu pri hodnotení úloh.

Píšte kód **konzistentne** a snažte sa dodržiavať štýl v ukážkach.

História: Larry Wall

- *1954, Los Angeles
- ‹`patch(1)`›, Perl
- Dvojnásobný víťaz *IOCCC*.
- Lingvista
- Silný veriaci (‹`bless`›, *apokalypsy*)

História: Pe(a)rl 1 (1987)

Perl officially stands for Practical Extraction and Report Language,
except when it doesn't.
...
Perl actually stands for Pathologically Eclectic Rubbish Lister,
but don't tell anyone I said that.
— man perl

História: Pe(a)rl 1 (1987)

- Shell
  ‹`$var`›, ‹`-f`›, ‹`-t`›, *heredocs*
- C
  syntax výrazov, konštrukcie
- Sed
  ‹`m//`›, ‹`s///`›, ‹`qr//`›
- … a ďalšie

História: Perl 5 (1994)

- Objekty
- Moduly

Najnovšia verzia `v5.36`, dostupná na Aise a Nymfe v moduloch

```
module add perl-5.36.0
```

Ak na svojom počítači nemáte Perl 5.36 v repozitári distribúcie, skúste `perlbrew`.

Slepá vetva: Perl 6 (2015), Raku (2019)

- Úplne nový a samostatný jazyk
- S Perlom už nemá okrem nepríčetnej syntaxi nič spoločné

Budúcnosť: Perl 7 (?)

- Plánovaný nástupca Perl 5
- Zahodenie veľmi starých a nepotrebných častí
- Stabilizácia nových vlastností (napr. signatúry)

Základy, výrazy, príkazy

Stiahnite si materiály podľa návodu:
‹https://gitlab.fi.muni.cz/xlacko1/pb173-perl›

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

use v5.36;

say "Hello, World!";
```

Prípadne priamo z terminálu:

```
$ perl -Mv5.36 -E 'say "Hello, World!"'
```

```
say VALUE, ...;
```

- Vypíše hodnoty v parametroch a ukončí riadok.

```
print VALUE, ...;
```

- Vypíše hodnoty v parametroch bez ukončenia riadka.
- ‹`say A, B, C`› ≈ ‹`print A, B, C, "\n"`›

```
printf FORMAT, ...;
```

- Vypíše formátovaný reťazec.
- Formátovacie značky ako ‹`printf(3)`›, a navyše nejaké ďalšie.

Pripomeňte si formátovací výpis.

Vyskúšajte si: `ex02p-name.pl`

Pohodlnejšiu *interpoláciu reťazcov* si ukážeme neskôr.

Číselné konštanty a operátory fungujú **skoro** ako v C:

```
use v5.36;

say 42;
say 1 + 2;
say 1 + 2 * 3;
say 5 ** 2;
say 7 % 3;
say 1 << 7;

say 3.1415;
say 1 / 2;
```

Základy

Jediné operátory z C, ktoré Perl nemá, sú

- `*` a `&`
  Perl má referencie
- `(T)` pre nejaký typ `T`
  V Perli nemá úplne zmysel

Reťazce sa spájajú operátorom ‹ . ›:

```
say "Hello" . ", World" . "!";
```

Základy: Kontext operátorov

Prečo má Perl ‹+› a ‹.›?

- ‹+›, ‹-› atď. sú **aritmetické** operátory
- ‹.› je **reťazcový** operátor

Tieto operátory dávajú hodnotám v operandoch **kontext**.

Čo sa stane, ak zmiešame typy operandov?

```
say "1" +  2;
say "1" + "2";
say  3  .  4;
say  3  . "4";
```

Základy: Kontext operátorov

Ako sa správajú aritmetické operácie na ľubovoľných reťazcoch?

```
say 3 + "";
say 7 - "3 days";
say 2 * "Rust";
say 4 / "OMG Perl";
```

Reťazec, číslo... prečo nie oboje?

- Skalárne premenné udržujú číselnú *aj* reťazcovú hodnotu.
- Niektoré operátory vynútia koerciu svojich argumentov.
- Reťazce sa prevádzajú na čísla mechanizmom ako `strtol(3)`.
- Perl varuje pri prevode reťazca, ktorý nereprezentuje číslo.

Desatinné čísla

Prevod do IEEE 754 je typicky bezstratový pre

- celé čísla,
- čísla, ktoré sa dajú vyjadriť ako súčet (potenciálne záporných) mocnín 2: $1.625 = 1 + \frac{1}{2} + \frac{1}{8} = 2^0 + 2^{-1} + 2^{-3}$

Pre ostatné čísla sa nemusí hodnota uložiť presne!

```
$ perl -E 'say 0.1 * 0.1 * 0.1 * 0.1 == 1/1000 ? "Yes" : "No";'
No
```

Základy: Zátvorky

Zátvorky vo výrazoch vynucujú prioritu vyhodnotenia:

```
1 + 2 * 3       # 7
(1 + 2) * 3     # 9
```

Niektoré jazyky ich tiež *vyžadujú* pri volaní funkcií:

```
fprintf(stderr, "memgrind: Invalid pointer\n");
```

Vyskúšajte si, ako sa správa Perl:

‹ex06p-parenthesis.pl›

Základy: Zátvorky

Je správanie príkladu ‹ex06p-parenthesis.pl› divné?

```
int pow(int n);
// ...
pow (1 + 2) * 3;
```

Konvencia

- *Zabudované* funkcie by sa mali volať bez zátvoriek.
- Pri jednoduchých výrazoch to zlepšuje prehľadnosť.

Zátvorky však píšte, ak je výraz v argumente zložitejší a nedá sa napísať inak, alebo ak si nie ste istí poradím vyhodnocovania.

Základy: Dátové typy

Perl má tri základné „typy“ premenných:

- `$NAME` Skaláry (čísla, reťazce, *referencie*, ...)
- `@NAME` Polia
- `%NAME` Asociatívne polia („hashe“)

`$`, `@` a `%` a volajú *sigils*.

Existujú ešte `&` a `*`, ktoré sa používajú oveľa menej.

Základy: Skaláry

Premenné deklarujeme pomocou `my`.
Pred názvom *skalárnej* premennej uvádzame `$`.

```
my $name = "Perl";
my $version = 5;

say $name, ": ", $version;
```

`my` zavádza *lexikálnu* premennú, ktorá žije od momentu deklarácie až do konca rozsahu.

Základy: Skaláry

Môžeme deklarovať aj viac premenných naraz, sú však nutné zátvorky (ide totiž o zoznamy, uvidíme nabudúce):

```
my ($a, $b) = (0, 1);
```

Čo sa stane, ak zátvorky vynecháme?

Nedefinovaná hodnota

```
my $result;
my $result = undef;
```

- Špeciálna hodnota `undef` (≈ `None` v Pythone).
- Predikátový operátor `defined EXPRESSION`

`undef` je v skutočnosti operátor, ktorý môže mať argumenty:

```
undef LVALUE
LVALUE = undef
```

Základy: Interpolácia

Namiesto

```
my $message = "Hello, " . $name . "!";
```

môžeme písať

```
my $message = "Hello, $name!";
my $message = "Hello, ${name}!";
```

Čo vypíše nasledujúci príklad?

```
my $base = "source";
say "$base_name.txt";
```

Aritmetické operátory

‹+A›	‹-A›	‹A += B›
‹A + B›	‹A - B›	‹A -= B›
‹A * B›	‹A / B›	‹A *= B›
‹A % B›	‹A ** B›	‹A /= B›
‹A++›	‹++A›	‹A %= B›
‹A--›	‹--A›	‹A **= B›

Bitové operátory

`~A`	`A & B`	`A | B`
`A ^ B`	`A << B`	`A >> B`

Vrátane skrátených, napr. `A &= B`

Reťazcové operátory

`S . T`

Ternárny operátor

`CONDITION ? IF_TRUE : IF_FALSE`

Koalescencia

`A // B` ≈ `defined A ? A : B`

Logické hodnoty

- Perl nemá konštanty `true` ani `false`
- Nepravdivé hodnoty: `undef`, `""`, `0` (≈ `"0"`), `()`
- Všetky ostatné hodnoty sú pravdivé

Aká je logická hodnota `0.0`, `0E0` alebo `"0 but true"`?

Experimentálne logické hodnoty v 5.36

`use experimental 'builtin'`, viď `perldoc builtin`

Logické operátory

`!`	`&&`	`||`
`not`	`and`	`or`

`||` a `&&` vracajú hodnotu posledného vyhodnoteného výrazu.
Na výber prvej definovanej možnosti však preferujte `//` (koalescencia).

> **!** Vo výrazoch preferujte `!`, `&&`, `||` pred `not`, `and` a `or`, pretože tie prvé majú vyššiu prioritu (ako v C).

Logické operátory

Aký je rozdiel v priorite operátorov ‹&&› a ‹and›?

```
say 1 &&  2;    # say(1 && 2)
say 1 and 2;    # say(1) and 2
```

Kedy je užitočná nižšia priorita?

```
open my $file, "<", $filename
    or die "$filename: $!";
```

Relačné operátory

Čísla	`<`	`>`	`<=`	`>=`	`==`	`!=`
Reťazce	`lt`	`gt`	`le`	`ge`	`eq`	`ne`

Porovnania je možné reťaziť, napr. `0 < $var < 10`

Komparátory

Číselný	`<=>`
Reťazcový	`cmp`

Priradenie

`‹LVALUE = EXPR›`

Priradenie je možné reťaziť ako v C: `‹$x = $y = 1›`.
Na rozdiel od C však vracia lvalue (čo to znamená?)

Operátor môže priradiť aj zoznamy:

```
($x, $y) = ($y, $x);
```

Zoznamy a polia budú vysvetlené neskôr.

Podmienky

```
if (CONDITION) {
    STATEMENTS...;
} elsif (CONDITION) {
    STATEMENTS...;
} else {
    STATEMENTS...;
}
```

Zátvorky ‹()› aj ‹{}› sú povinné.
Časti ‹elsif› aj ‹else› sú nepovinné.

Podmienky

```
unless (CONDITION) {
    STATEMENTS...;
}
```

Ekvivalent ‹`if`› s negáciou podmienky.
Radšej nepoužívajte, preferujte postfixový variant (neskôr).

```
unless ($a && ($b || !$c)) {
    STATEMENTS...;
} else {
    # … /o\
}
```

Cykly

```
while (CONDITION) {
    STATEMENTS...;
}
```

```
until (CONDITION) {
    STATEMENTS...;
}
```

Cykly

V tele cyklu môžeme použiť kľúčové slová:

- `last` ukončí cyklus (≈ `break` v C, Python)
- `next` ďalšia iterácia (≈ `continue` v C, Python)
- `redo` spustí tú istú iteráciu znova (zriedka používané)

Cykly

Za cyklom môže byť `continue BLOCK`, ktorý sa vykoná pred každou iteráciou.
Toto **nie** je `for/else` z Pythonu!

```
while ($i < $target) {
    next if !viable($i);        # Whoops, skipping increment…
    return $i if inspect($i);
} continue {                    # … but <continue> executes anyway
    $i++;                       # and increments the value.
}
```

Cykly

```
for (EXPRESSION; CONDITION; INCREMENT) {
    STATEMENTS...;
}
```

```
for VARIABLE (LIST) {
    STATEMENTS...;
}
```

Pre `for` existuje synonymum `foreach`.

Základy: Príkazy

Pred cyklom môžeme vytvoriť návestie ‹LABEL:›.
Na to potom môže odkazovať ‹last›, ‹next› aj ‹redo›.

```
OUTER: while ($i < $max) {
    while ($j < $max) {
        last OUTER if $i + $j == $search;
        # …
    }
}
```

Funkcie

Funkcie

Perl volá funkcie „subrutiny“, preto `sub`:

```
sub NAME(ARGUMENTS…) {
    STATEMENTS...;
}
```

Návratová hodnota

- *Explicitne* ‹`return EXPRESSION`› (**preferujte**)
- *Implicitne* hodnota posledného výrazu

```
sub add($a, $b) { return $a + $b; }
sub add($a, $b) { $a + $b }          # Same thing
```

Implicitne ukončujte **jedine** ak je telo funkcie jeden výraz,
alebo sa vrátená hodnota nemá používať (tj. ‹`void`› funkcia).

‹`return`› bez hodnoty vráti prázdny zoznam.

Východzie hodnoty

Parametre môžu mať východzie hodnoty.

```
sub greet($whom, $greeting = "Hello") {
    say "$how, $whom!";
}

say greet("World");     # Hello, World!
```

Špeciality

Postfixové podmienky

```
STATEMENT if CONDITION;
STATEMENT unless CONDITION;
```

- Používajte **veľmi** opatrne.
- ‹STATEMENT› aj ‹CONDITION› by mali byť veľmi jednoduché, dlhé podmienky napíšte civilizovane.

Vhodné na ošetrenie vstupných podmienok (*Early Return*).

Špeciality: Postfixové konštrukcie

```
sub factorial($n) {
    return 1 if $n <= 1;
    return $n * factorial($n);
}
```

Postfixové cykly

```
STATEMENT for LIST;              # Same with 'foreach'
STATEMENT while CONDITION;
STATEMENT until CONDITION;
```

Hodnotu vo `for` cykle získame z **implicitnej premennej** `$_`.

V týchto cykloch nie je možné použiť `last`, `next` ani `redo`.

Postfixový cyklus pre `for (A; B; C)` neexistuje.

Špeciality: Switch

Od ‹v5.36› dostupné už len ako experimentálna funkcionalita

```
use experimental 'switch';

for (EXPRESSION) {          # Or ‹given (EXPRESSION)›
    when (MATCH) { STATEMENT; }
    STATEMENT when MATCH;

    default { STATEMENT; }
}
```

Čo je ‹MATCH›? ‹perldoc perlsyn›

Špeciality: ‹do› blok

```
do {
    STATEMENTS...;
};
```

Vráti hodnotu posledného príkazu v bloku.
Možné spojiť s postfixovým ‹while› a ‹until›:

```
do {
    STATEMENTS...;
} while CONDITION;
```

Toto je stále *postfixový* cyklus, takže nie je možné použiť ‹last›, ‹next› ani ‹redo›!

Špeciality: Nepomenované argumenty

Ak funkcia musí mať argument, ale nepotrebujeme ho, nemusíme ho pomenovať.

```
sub run($command, $) {}
```

Pre argument s východzím parametrom nepotrebujeme ani hodnotu.
Musíme však Perlu naznačiť, že argument je nepovinný, kvôli arite.

```
sub run($command, $=) {}
```

V iných jazykoch sa často používa `_`.
Prečo nemôže mať Perl tiež `_` alebo `$_`?