perlintro

perlintro -- Trumpas įvadas į Perlą

Kas yra Perl?

Perl yra bendros paskirties programavimo kalba, kuri pradžioje buvo skirta manipuliavimui tekstu, o dabar yra naudojama daugeliui užduočių: sistemų administravimui, interneto svetainių, tinklo, grafinių sąsajų programavimui ir kitoms užduotims.

Ši kalba yra labiau praktiška (lengva naudoti, efektyvi, pilna) negu graži (maža, elegantiška, minimalistinė). Pagrindinės Perlo ypatybės yra naudojimo paprastumas, tiek procedūrinio, tiek objektinio programavimo palaikymas, patogus teksto apdorojimas ir vienas didžiausių nemokamų modulių pasirinkimas.

Perlo programų vykdymas

Norint vykdyti Perlo programą Unix komandinėje eilutėje tereikia parašyti:

```
perl programosvardas.pl
```

Arba galima parašyti tokią pirmąją programos eilutę:

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

... ir tada paleisti programą šitaip:

```
/kelias/iki/perl/programos.pl
```

Tiesa, ši programa turi turėti ``vykdymo'' nuostatą, tad reikia Unix parašyti

```
chmod 755 programa.pl
```

prieš ją paleidžiant.

Daugiau informacijos apie Perl interpretatorių galima rasti *perlrun* dokumentacijoje.

Paprasčiausios sintaksės apžvalga

Perlo programa susideda iš vieno ar kelių sakinių. Šie sakiniai paprastai būna surašyti byloje. Priešingai nei kai kuriose kitose kalbose, nereikia aprašyti `main()` funkcijos ar panašių dalykų.

Perlo sakiniai baigiasi kabliataškiu:

```
print "Sveikas, pasauli";
```

Komentarai prasideda grotelėmis ir tęsiasi iki eilutės pabaigos:

```
# Čia yra komentaras
```

Tarpai, naujos eilutės (*whitespace*) nieko nereiškia:

```
print
    "Sveikas, pasauli"
    ;
```

...nebent jie yra simbolių eilutėse:

```
# atspausdins "sveikas pasauli" dviejose eilutėse
print "Sveikas
pasauli";
```

Simbolių eilutes galima aprašyti tiek viengubomis, tiek dvigubomis kabutėmis:

```
print 'Sveikas, pasauli';
print "Sveikas, pasauli";
```

Tačiau tik dvigubose kabutėse esančios simbolių eilutės ``interpoliuoja" kintamuosius ir specialiuosius simbolius, tokius kaip naujos eilutės simbolis (`\n`):

```
print "Sveikas, $vardas\n"; # veikia kaip derėtų
print 'Sveikas, $vardas\n'; # spausdins $vardas\n taip kaip
parašyta
```

Skaičių nereikia imti į kabutes:

```
print 42;
```

Priklausomai nuo jūsų skonio, funkcijų argumentus galima rašyti skliausteliuose, o galima juos ir praleisti. Skliausteliai reikalingi tik retkarčiais norint paaiškinti operacijų eiliškumą.

```
print("Sveikas, pasauli\n");
print "Sveikas, pasauli\n";
```

Daugiau informacijos apie Perl sintaksę galima rasti *perlsyn* dokumentacijoje.

Perlo kintamųjų tipai

Perlas turi tris pagrindinius kintamųjų tipus: skaliarus, paprastuosius masyvus bei asociatyviuosius masyvus.

Skaliarai

Skaliarai saugo vieną reikšmę:

```
    my $gyvunas = "kupranugaris";
    my $atsakymas = 42;
```

Skaliarinės reikšmės gali būti simbolių eilutės, sveiki skaičiai arba skaičiai su kableliu. Perlas automatiškai pagal poreikį pavers vieną tipą į kitą. Nebūtina prieš vartojant kintamuosius juos deklaruoti.

Skaliarus galima naudoti įvairiai:

```
    print $gyvunas;
    print "Gyvūnas yra $gyvunas\n";
    print "$atsakymas pakėlus kvadratu gausime ", $atsakymas *
$atsakymas, "\n";
```

Perle yra keletas labai neįskaitomai atrodančių ``magiškų" skaliarinių kintamųjų. Šie specialūs kintamieji naudojami įvairiausiais tikslais ir visi yra aprašyti *perlvar* dokumentacijoje. Kol kas vienintelis toks specialus kintamasis, apie kurį vertėtų žinoti yra `$_`, kuris reiškia ``kintamąjį pagal nutylėjimą". Jis naudojamas daugelyje Perlo funkcijų, jei joms neperduodami kiti kintamieji, bei automatiškai nustatomas kai kuriuose cikluose.

```
    print; # pagal nutylėjimą atspausdina tai kas saugoma $_
```

Paprastieji masyvai

Paprastuose masyvuose saugomas reikšmių sąrašas:

```
    my @gyvunai = ("kupranugaris", "lama", "pelėda");
    my @skaiciai = (23, 42, 69);
    my @maisyta = ("kupranugaris", 42, 1.23);
```

Masyvų indeksai prasideda nuo nulio. Štai kaip prieinama prie masyvo elemento:

```
    print $gyvunai[0]; # spausdina "kupranugaris"
    print $gyvunai[1]; # spausdina "lama"
```

Specialusis kintamasis `$#masyvas` paskutinio masyvo elemento indekso numerį:

```
    print $maisyta[$#maisyta]; # spausdina paskutinį elementą -- 1.23
```

Galbūt jums norint sužinoti kiek yra masyve elementų norisi naudoti `$#masyvas + 1`, bet tai nėra būtina, nes naudojant `@masyvas` ten kur Perlas tikisi skaliaro (``skaliariniame kontekste") bus grąžinamas masyvo elementų skaičius:

```
    if (@gyvunai < 5) { ... } # jei masyve @gyvunai mažiau nei 5
elementai...
```

Norint gauti keletą reikšmių iš masyvo:

```
    @gyvunai[0,1];          # grąžina ("kupranugaris", "lama");
    @gyvunai[0..2];         # grąžina ("kupranugaris", "lama",
"pelėda");
    @gyvunai[1..$#gyvunai]; # grąžina visus elementus, išskyrus pirmą
```

Tai vadinama ``masyvo dalimi'' (*array slice*)

Su masyvais galima atlikti įvairius naudingus veiksmus:

```
    my @surikiuoti = sort @gyvunai;
    my @atgaline_tvarka = reverse @skaiciai;
```

Perle yra ir keletas specialiųjų masyvų: `@ARGV` (programai perduoti komandinės eilutės argumentai), `@_` (argumentai perduoti subrutinai). Visi jie aprašyti *perlvar* dokumentacijoje.

Asociatyvieji masyvai

Asociatyvus masyvas aprašo vardo-reikšmės porų aibę:

```
    my %vaisiu_spalvos = ("obuolys", "raudonas", "bananas",
"geltonas");
```

Naudojant tarpus ir `=>` operatorių galima perrašyti įskaitomiau:

```
    my %vaisiu_spalvos = (
        obuolys => "raudonas",
        bananas => "geltonas",
    );
```

Pasiekti elementus galima taip:

```
    $vaisiu_spalvos{"obuolys"}; # grąžina "raudonas"
```

Raktų bei reikšmių sąrašus galima gauti per funkcijas `keys()` ir `values()`

```
    my @vaisiai = keys %vaisiu_spalvos;
    my @spalvos = values %vaisiu_spalvos;
```

Asociatyvūs masyvai neturi kokios nors rikiavimo tvarkos, nors visada galima surikiuoti masyvą, kuris gaunamas su funkcija `keys()`.

Lygiai kaip yra specialiųjų skaliarų ar paprastų masyvų, taip yra ir specialių asociatyvių masyvų. Labiausiai naudojamas yra `%ENV` specialusis asociatyvusis masyvas, kuriame saugomi aplinkos kintamieji. Apie tai plačiau *perlvar* dokumentacijoje.

Plačiau apie skaliarus, paprastus ir asociatyviuosius masyvus galima pasiskaityti *perldata* dokumentacijoje.

Sudėtingesnės duomenų struktūros gali būti sudaromos naudojantis nuorodomis (*references*). Jų dėka galima sukurti sąrašus ir masyvus kituose sąrašuose bei masyvuose.

Nuoroda yra skaliarinė reikšmė, galinti rodyti į bet kurio tipo Perlo duomenis. Taigi, išsaugant nuorodą į masyvą kaip kito masyvo elementą, galima sudaryti masyvų

masyvus (daugiamačius masyvus). Štai pavyzdys, kuriame saugoma dviejų lygių asociatyvaus masyvo asociatyviame masyve struktūra pasinaudojus anoniminėmis nuorodomis:

```
    my $kintamieji = {
        skaliarai    => {
            apibudinimas => "viena reikšmė",
            simbolis     => '$',
            },
        masyvai      => {
            apibudinimas => "reikšmių sąrašas",
            simbolis     => '@',
            },
        asociatyvus  => {
            apibudinimas => "rakto/reikšmės poros",
            simbolis     => '%',
            },
    };
    print "Skaliarai prasideda simboliu ".
        "$kintamieji->{'skaliarai'}->{'simbolis'}\n";
```

Pilna dokumentacija apie nuorodas yra *perlreftut*, *perllol*, *perlref* ir *perldsc* dokumentacijose.

Kintamųjų sritys

Kol kas visada kintamuosius aprašėme naudodami šią sintaksę:

```
    my $kintamasis = "reikšmė";
```

Tiesą sakant, `my` nėra būtinas, galima tiesiog rašyti:

```
    $kintamasis = "reikšmė";
```

Tačiau jei praleidžiate `my`, sukuriamas globalusis kintamasis visoje jūsų programoje, o tai nėra pats geriausias programavimo būdas. `my` sukuria *leksinės srities* (*lexically scoped*) kintamąjį, kuris galioja tik tame bloke, kuriame jis yra aprašytas (blokas yra keletas sakinių, apskliaustų figūriniais skliaustais).

```
    my $a = "foo";
    if ($kazkas) {
        my $b = "bar";
        print $a; # spausdina "foo"
        print $b; # spausdina "bar"
    }
    print $a; # spausdina "foo"
    print $b; # nieko nespausdina, nes $b galiojimo blokas jau
baigėsi
```

Jeigu naudosite `my` kartu su `use strict;` jūsų programos pradžioje, Perlo interpretatorius galės pastebėti dažnas programuotojų klaidas ir apie tai jus įspėti. Tarkim aukščiau duotame pavyzdyje paskutinis `print $b;` išmestų klaidą ir programa neveiktų. Rekomenduojama visada naudoti `strict` sintaksę.

Sąlygos ir ciklų sakiniai

Perlas turi visus standartinius ciklos ir sąlygos sakinius išskyrus `case/switch` (bet jei jums tikrai jų reikia, galite pasinaudoti moduliu `Switch`, kuris netgi gali daugiau nei įprasti `switch/case` sakiniai kitose kalbose).

Sąlyga gali būti bet kuris Perlo sakinys. Kitame skyriuje bus aprašomi operatoriai, tad žiūrėkite ten, kokie sąlygos, Būlio logikos operatoriai dažnai naudojami sąlygos sakiniuose.

if

```
if ( sąlyga ) {
    ...
} elsif ( kita sąlyga ) {
    ...
} else {
   ...
}
```

Yra ir atvirkštinė versija:

```
unless ( sąlyga ) {
   ...
}
```

... kuri reiškia tą patį kaip ir `if (!sąlyga) { ... }`, tik `unless` lengviau perskaityti.

Perlo sąlygos sakiniuose figūriniai skliaustai yra būtini, net jei yra tik vienas sakinys sąlygos bloke. Tačiau galima be to apsieiti ir iškelti sąlygą į sakinio galą:

```
# tradicinis būdas
if ( $kazkas ) {
    print "aha!";
}
# labiau perliškas būdas:
print "aha!" if $kazkas;
print "nebėra bananų" unless $bananai;
```

while

```
while ( sąlyga ) {
    ...
}
```

Kaip ir su `unless`, yra ir atvirkštinė versija:

```
until ( sąlyga ) { # tas pats kaip while (!sąlyga)
    ...
}
```

`while` galima permesti ir į galą:

```
print "la la la\n" while 1; # amžinas ciklas
```

for

Lygiai taip kaip ir C:

```
for ($i = 0; $i <= $max; $i++) {
    ...
}
```

C stiliaus `for` ciklas retai naudojamas Perle, nes Perl turi draugiškesnį ir lengviau panaudojamą `foreach` ciklą.

foreach

```
foreach (@masyvas) {
    print "Masyvo elementas $_\n";
}
# nebūtina naudot $_...
foreach my $raktas (keys %hash) {
    print "Rakto $raktas reikšmė yra $hash{$raktas}\n";
}
```

Daugiau apie ciklo sakinius (ir dar apie tuos kurie čia nepaminėti) galima rasti *perlsyn* dokumentacijoje.

Operatoriai ir funkcijos

Perlas turi daug standartinių funkcijų. kai kurias jau matėme šiame įvade (tokias kaip `print`, `sort` arba `reverse`). Pilnas funkcijų sąrašas yra *perlfunc* dokumentacijoje ir apie kurią nors funkciją lengva pasiskaityti konsolėje įvedus `perldoc -f funkcijospavadinimas`.

Perlo operatoriai pilnai aprašyti *perlop* dokumentacijoje. Štai keletas dažniau sutinkamų:

Aritmetiniai:

```
+   sudėtis
-   atimtis
*   daugyba
/   dalyba
```

Matematinio palyginimo:

```
==   lygybės
!=   nelygybės
<    mažiau negu
>    daugiau negu
<=   mažiau arba lygu
>=   daugiau arba lygu
```

Simbolių eilučių palyginimo:

```
eq   lygybės
ne   nelygybės
lt   mažiau nei
gt   daugiau nei
le   mažiau arba lygu
ge   daugiau arba lygu
```

Kodėl reikia skirtingų palyginimo operatorių simbolių eilutėms ir matematinėms išraiškoms? Kadangi Perlas netipizuoja kintamųjų pagal tai ar tai skaičius, ar simbolių eilutė, jam reikia nurodyti ar rikiuoti matematiškai (kur 99 yra mažiau nei 100) ar alfabetiškai (kur 100 eina prieš 99)

Loginiai operatoriai:

```
&&   ir
||   ar
!    ne</pre>
```

`&&`, `||` bei `!` galima užrašyti ir `and`, `or`, `not`. Taip jie labiau įskaitomi, tačiau keičiasi pirmumo eilė. Plačiau apie skirtumus tarp `and` ir `&&` galima rasti *perlop* dokumentacijoje.

Kiti operatoriai:

```
=     priskyrimas
.     simbolių eilučių sujungimas
x     simbolių eilučių daugyba
..    intervalo operatorius (sukuria skaičių sąrašą)
```

Dauguma operatorių gali būti derinami su = šitokiu būdu:

```
$a += 1;    # tas pats kaip $a = $a + 1;
$a -= 1;    # tas pats kaip $a = $a - 1;
$a .= "\n"; # tas pats kaip $a = $a . "\n";
```

Bylos ir įvestis/išvestis

Įvesčiai ar išvesčiai bylą galima atidaryti su funkcija `open()`. Pilnai su visomis detalėmis ji aprašyta *perlfunc* bei *perlopentut* dokumentacijoje, tačiau trumpai:

```
    open(INFILE,  "infile.txt")    or die "Negaliu atidaryt
input.txt: $!";
    open(OUTFILE, ">outfile.txt")  or die "Negaliu atidaryt
outfile.txt: $!";
    open(LOGFILE, ">>logfile.txt") or die "Negaliu atidaryt
logfile.txt: $!";
```

Skaityti iš bylos galima naudojantis <> operatorių. Skaliariniame kontekste jis nuskaito vieną eilutę iš bylos, o sąrašo kontekste grąžina iš bylos eilučių sudarytą masyvą:

```
my $eilute = <INFILE>;
my @eilutes = <INFILE>;
```

Visos bylos nuskaitymas iš karto vadinamas ``šliurpimu" (*slurping*). Tai gali būti naudinga, bet kartu gali ir pareikalauti daug atminties resursų. Daugumą dalykų galima padaryti tekstą apdorojant po eilutę ir naudojantis Perlo ciklais.

Operatorius `<>` dažniausiai naudojamas tokiame `while` cikle:

```
    while (<INFILE>) { # priskiria kiekvieną bylos eilutę $_
        print "Ką tik perskaičiau eilutę: $_";
    }
```

Mes jau matėme kaip spausdinti tekstą naudojantis `print()`. Tačiau `print()` galima nurodyti pirmu argumentu į kurią bylą spausdinti:

```
    print STDERR "Paskutinis perspėjimas\n";
    print OUTFILE $irasas;
    print LOGFILE $ivykis;
```

Kai baigiate dirbti su bylomis, jas reiktų uždaryti su funkcija `close()` (nors tiesą sakant, Perlas sutvarkys viską ką pridarėte, net jei ir pamiršote uždaryti bylą)

```
    close INFILE;
```

Įpraiškos

Perlas palaiko sudėtingą ir plačią įpraiškų sintaksę. Pilną jos aprašymą galima rasti *perlrequick*, *perlretut* ir kituose dokumentacijos skyriuose. Tačiau trumpai:

Paprastas tikrinimas

```
    if (/foo/)       { ... } # 'true' jei $_ yra simbolių seka 'foo'
    if ($a =~ /foo/) { ... } # 'true' jei $a yra simbolių seka 'foo'
```

Įpraiškų operatorius `//` aprašytas *perlop* dokumentacijoje. Jis pagal nutylėjimą įprašką taiko kintamajam `$_`, tačiau tai galima pakeisti, pasinaudojus operatoriumi `=~` (kurio aprašymą irgi galima rasti *perlop* dokumentacijoje).

Paprastas pakeitimas

```
    s/foo/bar/;           # pakeičia 'foo' į 'bar' kintamajame $_
    $a =~ s/foo/bar/;     # pakeičia 'foo' į 'bar' kintamajame $a
    $a =~ s/foo/bar/g;    # pakeičia VISUS 'foo' į 'bar' kintamajame
$a
```

Pakeitimo operatorius `s///` aprašytas *perlop* dokumentacijoje.

Sudėtingesnės įpraiškos

Įpraiškos nebūtinai sudaromos iš pastovių simbolių eilučių. Tiesą sakant, naudojantis sudėtingesnėmis įpraiškomis galite aprašyti tokius šablonus, kokius tik sugalvosite. Pilnai tai aprašyta *perlre* dokumentacijoje, o čia pateikiama tik trumpa lentelė:

```
    .               vienas bet koks simbolis
    \s              tarpai, naujos eilutės ir tabuliacijos simboliai
(I<whitespace>)
    \S              ne tarpai, ne naujos eilutės ir ne tabuliacija
(I<non-whitespace>)
    \d              skaitmuo (0-9)
    \D              ne skaitmuo
    \w              žodžių simbolis (a-z, A-Z, 0-9 ir _)
    \W              ne žodžių simbolis
    [aeiou]         atitinka vieną simbolį iš duotos aibės
    [^aeiou]        atitinka vieną simbolį ne iš duotosios aibės
    (foo|bar|baz)   atitinka vieną iš alternatyvų
    ^               eilutės pradžia
    $               eilutės pabaiga
```

Taip pat galima nurodyti kiek kartų įpraiška turi atitikti prieš tai nurodytą išraišką, kur ``išraiška'' yra paprastas simbolis arba vienas iš metasimbolių nurodytų prieš šią pastraipą esančioje lentelėje.

```
    *       nulį ar daugiau kartų
    +       vieną ar daugiau kartų
    ?       nulį arba vieną kartą
    {3}     lygiai tris kartus
    {3,6}   nuo trijų iki šešių kartų
    {3,}    tris ar daugiau kartų
```

Keletas trumpų pavyzdžių:

```
    /^\d+/          eilutė prasidedanti vienu ar daugiau skaitmenų
    /^$/            tuščia eilutė (po eilutės pradžios iš karto eina
eilutės pabaiga)
    /(\d\s){3}/     trys skaitmenys atskirti tarpais ar tabuliacija
(tarkim "3 4 5 ")
    /(a.)+/         eilutė, kurioje kiekviena neporinė raidė yra 'a'
(pvz 'abacadaf')
    # Šis ciklas skaito iš STDIN ir spausdina netuščias eilutes:
    while (<>) {
        next if /^$/;
        print;
    }
```

Skliausteliai

Skliausteliai naudojami ne vien tik grupavimui -- jie turi ir kitą paskirtį. Jie gali būti naudojami įpraiškos rezultatų išsaugojimui. Rezultatai atsiduria kintamuosiuose `$1`, `$2` ir taip toliau.

```
    # pigus ir ne visai tikslus būdas išskaidyti el.pašto adresą į
dalis:
    if ($email =~ /([^@])+@(.+)/) {
        print "Vartotojo vardas: $1\n";
        print "Domenas: $2\n";
    }
```

Kitos įpraiškų galimybės

Perlo įpraiškos dar palaiko daugybę dalykų (*backreferences, lookaheads* ir t.t.) Apie visą tai galima pasiskaityti *perlrequick*, *perlretut* ir *perlre* dokumentacijose.

Funkcijų rašymas

Labai lengva rašytis savo funkcijas:

```
sub log {
    my $pranesimas = shift;
    print LOGBYLA $pranesimas;
}
```

Ką veikia tas `shift`? Na, funkcijai perduodami argumentai atsiranda masyve `@_` (apie tai daugiau *perlvar* dokumentacijoje). Jeigu `shift` neperduodami jokie argumentai, ši funkcija naudoja masyvą `@_`. Tad `my $pranesimas = shift;` paima pirmą argumentų masyvo narį ir priskiria jį kintamajam `$pranesimas`.

Su `@_` galima elgtis ir kitaip:

```
my ($pranesimas, $svarbumas) = @_; # dažnas būdas
my $pranesimas = $_[0];            # retas ir bjaurus būdas
```

Funkcijos gali grąžinti reikšmes:

```
sub kvadratas {
    my $skaicius = shift;
    my $rezultatas = $skaicius * $skaicius;
    return $rezultatas;
}
```

Daugiau apie funkcijas *perlsub* dokumentacijoje

Objektinis Perlas

Objektinis Perlas yra ganėtinai paprastas. Objektai Perle yra nuorodos, kurios magiškai žino *kokį* objektą jos vaizduoja. Tačiau objektinis Perlas neįeina į šį trumpą įvadą. Skaitykite *perlboot*, *perltoot*, *perltooc* ir *perlobj* dokumentaciją.

Pradedantieji Perlo programuotojai dažniausiai su objektiniu programavimu susiduria tik naudodamiesi Perlo moduliais.

Naudojimasis Perlo moduliais

Perlo moduliai leidžia jums neišradinėti dviračio, nes galima naudotis jau kažkieno parašytu kodu. Modulius galima parsisiųsti iš http://www.cpan.org . Nemažai modulių yra įdiegiami kartu su pačiu Perlu.

Moduliai CPAN svetainėje yra kategorizuoti. Yra daug įvairiausių kategorijų: nuo teksto apdorojimo iki tinklo protokolų, nuo duomenų bazių iki grafikos.

Norint sužinoti kaip naudotis kuriuo nors moduliu, komandinėje eilutėje parašykite `perldoc Modulio::Pavadinimas`. Kode dažniausiai jums reikės rašyti `use Modulio::Pavadinimas` -- tai įkraus moulį ir leis naudotis jo funkcijomis.

perlfaq dokumentacijoje yra dažnai užduodami klausimai ir atsakymai apie dažnai pasitaikančias užduotis. Dažnai atsakymai siūlo naudotis vienu ar kitu moduliu.

perlmod dokumentacija bendrai aprašo Perlo modulius, *perlmodlib* dokumentacijoje yra sąrašas modulių, kurie įdiegti kartu su Perlu.

Jeigu patys norite rašytis Perlo modulius, skaitykite *perlnewmod* dokumentaciją.