Przetwarzanie tekstu 3-4

Perl

Perl

informacje ogólne

- autor: Larry Wall, 1987
- perl = **p**ractical **e**xtraction and **r**eport **l**anguage
- /perl = **p**athologically **e**clectic **r**ubbish **l**ister/
- zapożyczenia z C, sed, awk, sh
- zoptymalizowany do przetwarzania tekstu
- program jest interpretowany

Perl

uruchomienie programu

uruchomienie przez jawne wywołanie interpretera Perla

plik `p1.pl`

```
$nazwa="perl";
$wersja=5;
print $nazwa,$wersja,"\n";
```

polecenie: `perl p1.pl`

Perl

uruchomienie programu

uruchomienie jako pliku wykonywalnego

plik `p1.pl`

```
#!/usr/bin/perl

$nazwa="perl";
$wersja=5;
print $nazwa,$wersja,"\n";
```

polecenie: `p1.pl`
(wcześniej `chmod +x p1.pl`)

Perl

zmienne

- nazwa zmiennej w Perlu rozpoczyna się od przedrostka:

  $ @ % & *

  Znak ten określa typ wartości zmiennej:

typ	przedrostek
skalarny	$
tablica indeksowana	@
tablica asocjatywna (hasz)	%
funkcja	&
typ nieokreślony	*

- każda zmienna rozpoczyna się od jednego z tych znaków (wyjątek: w wywołaniach funkcji można zwykle pominąć znak &)

Perl

deklaracje zmiennych

- zmienne o zasięgu leksykalnym: zakresem jest blok (`{ ... }`)

```
my $a;
my @b;
my %c;
```

- zmienne o zasięgu globalnym (zmienne pakietowe)

```
our $d;
our %e;
local $f;
local @g;
```

- zmiennych można używać bez uprzedniego zadeklarowania, niezadeklarowana zmienna jest zmienną o zasięgu globalnym

Perl

wartości skalarne

- wartości skalarne to:
  - liczby: `1  1.5  1.5e2`
  - napisy: `'Dzień dobry'  "Dzień dobry"`
  - odwołania ($\approx$ wskaźniki, adresy): `\$b  \%h`
- zmiennej skalarnej można w każdej chwili przypisać dowolną wartości skalarną
- poprawny jest następujący fragment kodu:

```
my $a="perl";
$a=5;
$a=\$b;
```

Perl

napisy pojedynczo i podwójnie cytowane

- w napisach podwójnie cytowanych ma miejsce interpolacja zmiennych i sekwencji specjalnych `\n \t ...`
- w napisach pojedynczo cytowanych – nie
- interpolacja zmiennych – zastąpienie nazwy zmiennej jej wartością

```
my $a="perl";
my $b=5;
print "$a wersja $b\n"     % perl wersja 5
print ’$a wersja $b\n’;    % $a wersja $b\n
```

Perl

tablice indeksowane

- indeksy są kolejnymi liczbami całkowitymi od 0
- tablica powiększa się dynamicznie wraz z dodawamiem nowych elementów
- elementy tablicy są dowolnymi wartościami skalarnymi
- odwołując się do elementu tablicy używamy $

```
my @t;          # deklarację można pominąć
my $t="dwa";    # @t i $t to różne zmienne
$t[0]=1;        # indeks w nawiasach kwadratowych
$t[1]=$t;       # przypisujemy wartość "dwa"
```

Perl

tablice asocjatywne

- klucze są napisami
- tablica powiększa się dynamicznie wraz z dodawaniem nowych elementów
- elementy są dowolnymi wartościami sklalarnymi

```
my %h;                    # deklarację można pominąć
$h{’styczeń’}=1;            # klucz w nawiasach klamrowych
$h{’luty’}=2;
print "styczeń to miesiąc $h{’styczeń’}";
```

Perl

kontekst

- każda zmienna czy wyrażenie ewaluowane jest zawsze w określonym kontekście
- to samo wyrażenie może zwracać różną wartość w zależności od kontekstu
- możliwe konteksty: kontekst skalarny i kontekst listy
- w kontekście skalarnym wyrażenie zwraca wartość skalarną, w kontekście listy – listę (ciąg wartości skalarnych)
- listę zapisujemy używając przecinków i nawiasów, np. `(1,2,3)`
- kontekst wyznaczają operatory, funkcje, instrukcje sterujące, a także zmienne, do których następuje przypisanie

```
@t=(11,22,33);
@v=@t;              # skopiowanie tablicy
$n=@t;              # $n==3 (wartością tablicy w kontekście
                    # skalarnym jest jej rozmiar)
print @t;           # 112233
print @t+1;         # 4
```

Perl

operatory (niekompletne)

- wszystkie operatory języka C (łącznie ze skrótami *operator*=), poza & i `*` w znaczeniu wkaźnika i dereferencji
- potęgowanie: `**` `**=`
- konkatenacja napisów: `.` `.=`
- porównanie napisów:
  `eq` `ne` `lt` `gt` `le` `ge` `cmp`
- potęgowanie (powtórzenie) napisów: `x` `x=`
- testowanie plików: `-e` (czy istnieje) `-f` (czy zwykły) `-d` (czy katalog) ...

Perl

wejście/wyjście: uchwyty plików

- predefiniowane: `STDIN` `STDOUT` `STDERR`
- utworzenie uchwytu pliku/potoku

`open(WE,"<plik");`	do odczytu
`open(WY,">plik");`	do zpisu
`open(WY,">>plik")`	do dopisywania
`open(WE,"polecenie |")`	do odczytu
`open(WY,"| polecenie")`	do zapisu

KONWENCJA: nazwy uchwytów plików piszemy wielkimi literami

Perl

wejście/wyjście: czytanie z pliku

- operator <>
- w kontekście skalarnym zwraca kolejną linię pliku (wraz ze znakiem końca linii) lub wartość fałsz, gdy koniec pliku
- w kontekście listy zwraca listę linii pliku wejściowego
- użyty bez argumentu – dotyczy standardowego wejścia lub plików podanych jako argumenty wywołania (patrz następny slajd)

```
open(WE,"plikwe.txt");

$linia=<WE>;      # odczytane kolejnej linii

@linie=<WE>;      # odczytanie wszystkich (pozostalych) linii
```

Perl

wejście/wyjście: operator **<>** bez argumentów

- **to jest najważniejszy slajd dotyczący wejścia!**
- operator <> jest najczęściej uzywany BEZ ARGUMENTU, wtedy:
  - jeśli tablica @ARGV nie jest pusta, program domniemuje, że zawiera ona nazwy plików wejściowych i czyta kolejno z tych plików
  - jeśli tablica @ARGV jest pusta, czyta za standardowego wejścia
- używając operatora <> trzeba pamiętać, aby na początku programu (zanim użyjemy <>) usunąć z tablicy @ARGV wszystkie ewentualne parametry programu, pozostawiając w niej tylko nazwy plików wejściowych
- w ten sposób uzyskuje się najbardziej standardowe (oczekiwane przez użytkownika) zachowanie programu
- jeśli nie ma powodu, żeby zrobić inaczej, PROGRAM POWINIEN CZYTAĆ Z <> !!! bo wtedy zachowuje sie tak, jak tego oczekuje uzytkownik (przyzwyczajony do pracy konsolowej pod Lunuxem/Unixem)

Perl

wejście/wyjście: zapis do pliku

- funkcje: `print` `printf`

```
open(WY,">plikwy.txt");

print WY "Dzień dobry.\n";        # do pliku plikwy.txt
print STDOUT "Dzień dobry.\n"      # na standardowe wyjście
print "Dzień dobry.\n";           # na standardowe wyjście

printf WY "%s %s.\n", "Dzień", "dobry";
```

- jeśli nie ma powodu, żeby zrobić inaczej, program powinien pisać na standardowe wyjście

Perl

instrukcje sterujące (podstawowe)

- `if ( `*wyr*` )` *BLOK*
- `if ( `*wyr*` )` *BLOK* `else` *BLOK*
- `if ( `*wyr*` )` *BLOK* `elsif` *BLOK* `else` *BLOK*
- `while ( `*wyr*` )` *BLOK*
- `do` *BLOK* `while` *wyr*
- `for ( `*wyr*` ; `*wyr*` ; `*wyr*` )` *BLOK*
- `for $`*zmienna* `(` *lista* `)` *BLOK*

BLOK – ciąg instrukcji ujęty w nawiasy klamrowe. Nawiasy są obowiązkowe.

Perl

instrukcje sterujące (jako modyfikatory wyrażeń)

- *INSTRUKCJA* `if` *wyr*
- *INSTRUKCJA* `while` *wyr*

Perl

instrukcje sterujące – warunki

- warunkiem może być dowolne wyrażenienie (każdy operator i każda funkcja zwraca jakąś wartość)
- fałsz: `""` `"0"` `0`
- pozostałe wartości traktowane jako prawda

Perl

przykład: iteracja po liniach pliku wejściowego – przykłady

program numerujący linie pliku – 3 wersje

```
my $nrlinii=0;
my $linia;
while(my $linia=<>)
{
  $nrlinii++;
print "$nrlinii: $linia";
}
```

```
while(<>)
{
  print ++$nrlinii . ': ' . $_;
}
```

```
print ++$nrlinii . ': ' . $_ while(<>)
```

Perl

iteracja po elementach tablicy indeksowanej – przykład

```
@tab=("styczeń","luty","marzec");
for($i=0;$i<@tab;$i++)
{
  print $tab[$i]."\n";
}
```

```
@tab=("styczeń","luty","marzec");
for $m (@tab)
{
  print $m."\n";
}
```

Perl

iteracja po elementach tablicy asocjatywnej – przykłady

```
hasz=("styczeń"=>1,"luty"=>2,"marzec"=>3);

for $klucz (keys %hasz)
{
  $linia="$klucz to miesiąc $hasz$klucz.\n";
  print $linia;
}
```

```
("styczeń"=>1,"luty"=>2,"marzec"=>3)
```

jest wariantem notacyjnym

```
("styczeń",1,"luty",2,"marzec",3)
```

Perl

wyrażenia regularne Perla

- sekwencje specjalne – klasy znaków:

  `\s` – biały znak
  `\S` – dopełnienie `\s`
  `\d` – cyfra
  `\D` – dopełnienie `\d`
  `\w` – znak wyrazu (litera, cyfra lub _)
  `\W` – dopełnienie `\w`

Perl

dopasowanie

- sprawdzenie czy w napisie występuje fragment pasujący do wzorca:

$$napis \ =\sim \ /wyr\_reg/$$

- sprawdzenie czy w napisie nie występuje fragment pasujący do wzorca:

$$napis \ !\sim \ /wyr\_reg/$$

```
print "Imię i nazwisko: ";
$in=<STDIN>;
chomp($in);
if($in !~ /\s+[A-Z][a-z]\s+[A-Z][a-z]+\s+$/)
{
  print "To nie wygląda jak imię i nazwisko.\n";
}
```

Perl

dopasowanie: przechwytywanie

- do przechwytywania fragmentów napisu dopasowanego do wzorca używamy nawiasów okrągłych
- do przechwyconych napisów odwołujemy się za pomocą zmiennych `$1, $2, ...`

```
print "Imię i nazwisko: ";
$in=<STDIN>;
chomp($in);
$in =~ /^\s+([A-Z][a-z]+)\s+([A-Z][a-z]+)\s+$/;
$imie=$1;
$nazwisko=$2;
```

Perl

podstawienie

- zastąpienie fragmentu napisu pasującego do wzorca innym napisem :

$$\$zmienna \ =\sim \ s/wyr\_reg/napis/$$
$$\$zmienna \ =\sim \ s/wyr\_reg/napis/g$$

```
$zdanie = "Ala ma kota.";
$zdanie =~ s/kota/psa/;   % $zdanie eq "Ala ma psa"
```

```
$zdanie = "     Ala    ma       kota    .     ";
$zdanie =~ s/^\s+//;   % usunięcie odstępów na początku
$zdanie =~ s/\s+$//;   % usunięcie odstępów na końcu
$zdanie =~ s/\s+/ /g;  % zastąpienie ciągów odstępów jednym
```

Perl

podstawienie

- we wzorcach i podstawianych napisach ma miejsce interpolacja zmiennych!

```
print "co: ";
$co=<>;
print "czym: ";
$czym=<>;
print "w czym: ";
$napis=<>;
$napis =~ s/$co/$czym/g;
print "\nwynik: $napis\n";
```

Perl

skróty

- operator <> bez argumentu czyta z plików podanych w linii wywołania lub, jeśli nie podano, ze standardowego wejścia (równoważne <ARGV>)
- $_ – domyśna zmienna, do której następuje przypisanie przeczytanej linii w pętli `while`, oraz na której wykonywane są m.in. operacje dopasowania, podstawienia, drukowania (`print`)

```
while(<>)
{
  s/([0-9]+)/$10/g;
  print;
}
```

równoważne

```
while($_=<ARGV>)
{
  $_ =~ s/([0-9]+)/$10/g;
  print $_;
}
```

Perl

operator dopasowania `m//` (uzupełnienie)

- pełna postać operatora dopasowania to `m//` (`m` można pominąć)
- w kontekście skalarnym zwraca: prawdę (1) lub fałsz (””)
- w kontekście listy zwraca listę podnapisów dopasowanych przez nawiasy przechwytujące; jeśli dopasowanie się nie powiedzie – listę pustą

```
$s="  Jan   Kowalski  ";
if(($im, $nazw) = $s =~ /^\s*([A-Z][a-z]+)\s+([A-Z][a-z]+)\s*$/)
{  ...  }
```

Perl

operator dopasowania `m//` z modyfikatorem g

- operator `m//g` szuka wszystkich dopasowań w napisie
- w kontekście listy zwraca listę podnapisów dopasowanych przez nawiasy przechwytujące lub – jeśli nie ma nawiasów – przez cały wzorzec
- w kontekście skalarnym przy kolejnych wywołaniach zwraca prawdę tyle razy ile razy dopasowanie się powiedzie, potem fałsz

```
$s=<>;
@liczby = $s =~ /[0-9]+/g;
print "wykryto liczby: ",join(',',@liczby),"\n";
```

```
$s=<>;
while($s =~ /([0-9]+)/g)  print "wykryto liczbę $1\n";
```

Perl

operator podstawienia `s///` (uzupełnienie)

- niezależnie od kontekstu zwraca liczbę wykonanych podstawień

```
while(<>)
{
  $n += s/([0-9]+)/<NUM>$1<NUM>/g;
}

print "liczba podstawień: $n\n";
```

Perl

przechwytywanie podnapisów (uzupełnienie)

- do podnapisu przechwyconego za pomocą nawiasów okrągłych odwołujemy się:
  - w tym samym wzorcu – za pomocą sekwencji \1, \2, ...
  - poza nim – za pomocą zmiennych $1, $2, ...

```
while(<>)
{
  if(/\b(\w+)\s+\1\b/)  print "Powtórzone słowo $1\n";
}
```

```
while(<>)
{
  while(/\b(\w+)\s+\1\b/g)  print "Powtórzone słowo $1\n";
}
```

Perl

testowanie lewego i prawego kontekstu dopasowania

- (*?=wzorzec*) – test prawego kontekstu
- (*?<=wzorzec*) – test lewego kontekstu
- przykład

```
$line=<>;
$line =~ s/\b(\w+)\s+(?=\1\b)//g;
print $line;
```

we: `usun usun powtarzajace sie sie sie slowa`
wy: `usun powtarzajace sie slowa`

Perl

obliczanie podstawienia: modyfikator e

- użycie modyfikatora e sprawia, że podstawiany napis jest obliczany: jest wynikiem ewaluacji wyrażenia będącego drugim argumentem operatora s
- przykład

```
$line=<>;
$line =~ s/(\d+)\s*\+\s*(\d+)/$1+$2/e;
print $line;
```

we: `Ala ma 1 + 2 koty.`
wy: `Ala ma 3 koty.`

Perl

kod w wyrażeniach regularnych: asercja (?{ *kod* })

- przykład

```
$line=<>;
$line =~ /(\d+)\s*\+\s*(\d+)(?{print "=".$1+$2;})/;
```

we: 1 + 2
wy: =3

Perl

wyrażenia regularne jako wartości zmiennych

- `qr//` – operator cytowania dla wyrażeń regularnych
- przykład

```
$co=qr/([A-Z][a-z]+)/;
$czym="Jola";
$s="Ala ma kota";
$s =~ s/$co/$czym/;
print $s;
```

Perl

operator transliteracji `tr///`

- działanie podobne do programu **`tr`**
- przykłady
  - zamiana wielkich liter na małe w napisie `$s`

```
$s =~ tr/[A-Z]/[a-z]/;
```

  - zamiana kodów polskich znaków z windows na Latin 2

```
while(<>)
{
  tr/\xB9\x9C\x9F/\xB1\xB6\xBC/;
  print;
}
```

Perl

funkcje

- definicja funkcji (najprostsza postać)

```
sub nazwa szablon
{
    instrukcje
}
```

- szablon opisuje typy argumentów, jest opcjonalny; jeśli wystąpi w definicji funkcji, wtedy funkcję można wywołać jak operator (bez nawiasów)
- argumenty nie mają nazw; lista argumentów wywołania dostępna jest wewnatrz funkcji jako wartość zmiennej @_
- wartością zwracaną jest wartość ostatniego obliczonego wyrażenia

Perl

funkcje: przykład – generowanie tekstu

genhtml.pl

```
sub html { $s=shift @_;
  "<HTML>$s\n</HTML>\n"; }

sub head { $s=shift @_;
  "\n<HEAD>$s\n</HEAD>"; }

sub title { $s=shift @_;
  "\n<TITLE>$s</TITLE>"; }

sub body { $s=shift @_;
  "\n<BODY>$s\n</BODY>"; }

sub ul   { while($item = shift @_) { $items.="\n<LI>$item"; }
  "\n<UL>$items\n</UL>"; }

@postacie=("żaby","ryby","raki");

print html(head(title("Obsada")).body(ul(@postacie)));
```

Perl

funkcje: przykład – generowanie tekstu – wyjście programu `genhtml.pl`

```
<HTML>
<HEAD>
<TITLE>Obsada</TITLE>
</HEAD>
<BODY>
<UL>
<LI>żaby
<LI>ryby
<LI>raki
</UL>
</BODY>
</HTML>
```

Perl

funkcje wbudowane – napisy (wybrane)

`chomp(`*napis*`)`	usunięcie znaku nowej linii z końca napisu
`split(/`*sep*`/,`*napis*`)`	podział napisu względem separatora *sep* (*sep* jest wyrażeniem regularnym)
`join(`*sep*`, `*lista*`)`	połączenie listy napisów w jeden z użyciem separatora *sep* (*sep* jest napisem)
`length(`*napis*`)`	długość napisu

Perl

funkcje wbudowane – tablice indeksowane (wybrane)

`pop(`*@tablica*`)`	pobranie elementu z końca
`push(`*@tablica*`,`*lista*`)`	dodanie elementów z listy na koniec tablicy
`shift(`*@tablica*`)`	pobranie elementu z początku tablicy
`unshift(`*@tablica*`,`*lista*`)`	dodanie elementów z listy na początku tablicy

Perl

funkcje wbudowane – tablice asocjatywne (wybrane)

`kyes(%hasz)`	zwraca listę kluczy
`values(%hasz)`	zwraca listę wartości
`delete($hasz{$klucz})`	usuwa element

Perl

funkcje wbudowane – listy (wybrane)

`reverse(`*lista*`)`	odwraca listę
`sort(`*lista*`)`	sortuje listę
`grep(`*wyrażenie*`,`*lista*`)`	zwraca listę złożoną z tych elementów listy *lista*, dla których wyrażenie zwraca wartość prawda (zamiast wyrażenia może być blok)
`map(`*wyrażenie*`,`*lista*`)`	zwraca listę wartości wyrażenia dla kolejnych elementów listy (zamiast wyrażenia może być blok)

Perl

pakiety

- pakiet = tablica symboli
- zawsze jeden pakiet jest pakietem aktualnym
- pakietem programu głównego jest `main`
- aktualny pakiet zmienia się deklaracją

```
package pakiet;
```

- zmienne deklarowane z użyciem słów `our` i `local` oraz wszystkie funkcje należą do pakietu (symbole globalne w obrębie pakietu)
- zmienne zadeklarowane z użyciem słowa `my` nie należą do pakietu (symbole lokalne – zakresem jest blok)

Perl

moduły

- moduł jest jednostką kodu perla przeznaczoną do wielokrotnego użytku
- przestrzeń symboli modułu zamyka się w pakiecie o tej samej nazwie, co nazwa modułu
- moduły występują w dwóch odmianach: tradycyjnej i obiektowej

Perl

moduły: przykład modułu tradycyjnego

TOHTML.pm

```
package TOHTML;
require Exporter;                                  #moduł obsługujący eksport symboli

our @ISA = (’Exporter’);
our @EXPORT = (’html’,’head’,’title’,’body’,’ul’,’$autor’); #symbole eksportowane

sub html  $cont=shift @_;
  "<HTML>\n$cont\n</HTML>\n";

sub head  $cont=shift @_;
  "\n<HEAD>$cont\n</HEAD>";

sub title  $title=shift @_;
  "\n<TITLE>$title</TITLE>";

sub body  $cont=shift @_;
  "\n<BODY>$cont\n</BODY>";

sub ul    while($item = shift @_)  $items.="\n<LI>$item";
  "\n<UL>$items\n</UL>"

our $autor="TO";
```

Perl

moduły: **use**

- przykład wykorzystania: funkcja **use**
- symbole z pakietu TOHTML wymienione w tablicy @EXPORT są importowane do pakietu aktualnego, w tym wypadku do pakietu main.
- Moduł TOHTML jest ładowany w czasie kompilacji

```
use TOHTML;

my @postacie=("żaby","ryby","raki");
print html(head(title("Obsada")).body(ul(@postacie)));

our $autor;
print "\n\nAutor generatora HTML:$autor\n";
```

Perl

moduły: require

- przykład wykorzystania: funkcja `require`

```
require TOHTML;   # Żadne symbole nie są dołączanie do pakietu main,
                  # dlatego nazwy zmiennych trzeba poprzedzac kwalifikatorem
                  # w postaci <nazwa_modułu>::
                  # Moduł ładowany w czasie wykonywania programu.

@TOHTML::postacie=("żaby","ryby","raki");

print TOHTML::html(
TOHTML::head(
TOHTML::title("Obsada")).
TOHTML::body(TOHTML::ul(@TOHTML::postacie)));
```

Perl

CPAN

- CPAN to podstawowe żródło internetowe związane z Perlem
- CPAN = Comprehensive Perl Archive Network
- dystrybucje perla
- dokumentacja
- tysiące modułów

Perl

najważniejsze tematy nieomówione

- odwołania
- różnica pomiędzy deklaracjami `our` i `local`
- wielowymiarowe struktury danych
- tablice anonimowe
- tworzenie modułów obiektowych

Perl

literatura

- L. Wall, T. Christiansen, J. Orwant, *Perl. Programowanie*, Wydawnictwo RM, Warszawa 2001