Narzędzia informatyczne w językoznawstwie

Perl - Manipulowanie tablic, zmienne lokalne, funkcje

Marcin Junczys-Dowmunt
junczys@amu.edu.pl

Zakład Logiki Stosowanej
http://www.logic.amu.edu.pl

12. grudnia 2007

Dzisiejszy wykład

- Wrócimy do tablic i omówimy:
  - wydajne dodawanie i usuwanie elementów z tablic

Dzisiejszy wykład

- Wrócimy do tablic i omówimy:
  - wydajne dodawanie i usuwanie elementów z tablic
- Omówimy zmienne lokalne oraz zakresy ważności zmiennych

Dzisiejszy wykład

- Wrócimy do tablic i omówimy:
  - wydajne dodawanie i usuwanie elementów z tablic
- Omówimy zmienne lokalne oraz zakresy ważności zmiennych
- Wprowadzimy pojęcie funkcji i omówimy:
  - wykonywanie funkcji
  - zwracanie argumentów przez funkcje
  - przekazywanie argumentów do funkcji

Usuwanie elementów z początku tablicy

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  print "Kasjerka obsługuje: ".shift(@kolejka)."\n";
}
```

Usuwanie elementów z początku tablicy

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  print "Kasjerka obsługuje: ".shift(@kolejka)."\n";
}
```

- Funkcja `shift` usuwa pierwszy element tablicy i zwraca go
- Indeksy "przesuwają" się, stąd nazwa *shift*

Dodawanie elementów do początku tablic

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca do okna\n";
    unshift(@kolejka, $aktualna);
  }
}
```

Dodawanie elementów do początku tablic

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca do okna\n";
    unshift(@kolejka, $aktualna);
  }
}
```

- Funkcja `unshift` wstawia element na początek tablicy
- Wstawia kilka elementów, gdy drugim elementem jest lista
- "Przesunięcie" indeksów następuje w drugą stronę

Dodawanie elementów do końca tablicy

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca na koniec\n";
    push(@kolejka, $aktualna);
  }
}
```

Dodawanie elementów do końca tablicy

```perl
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca na koniec\n";
    push(@kolejka, $aktualna);
  }
}
```

- Funkcja push wstawia element na koniec tablicy
- Wstawia kilka elementów, gdy drugim elementem jest lista
- Dodawanie elementów na końcu nie "przesuwa" indeksów

Usuwanie elementów z końca tablica

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca na koniec\n";
    push(@kolejka, $aktualna);
  }
  if(rand() < 0.5) {
    print pop(@kolejka)." ma dosyć czekania\n";
  }
}
```

Usuwanie elementów z końca tablica

```
@kolejka = qw(Marysia Janek Hania Kuba)
while(@kolejka) {
  print "Osoby w kolejce: ".join(@kolejka)."\n";
  $aktualna = shift(@kolejka);
  print "Kasjerka obsługuje: $aktualna\n";

  if(rand() < 0.5) {
    print "Zly bilet, $aktualna wraca na koniec\n";
    push(@kolejka, $aktualna);
  }
  if(rand() < 0.5) {
    print pop(@kolejka)." ma dosyć czekania\n";
  }
}
```

- Funkcję pop poznaliśmy już wcześniej

Wstawianie i usuwanie dowolnych elementów tablicy I

```
1  splice TABLICA,PRZESUNIĘCIE,DŁUGOŚĆ,LISTA
   splice TABLICA,PRZESUNIĘCIE,DŁUGOŚĆ
   splice TABLICA,PRZESUNIĘCIE
   splice TABLICA
```

Wstawianie i usuwanie dowolnych elementów tablicy I

```
splice TABLICA,PRZESUNIĘCIE,DŁUGOŚĆ,LISTA
splice TABLICA,PRZESUNIĘCIE,DŁUGOŚĆ
splice TABLICA,PRZESUNIĘCIE
splice TABLICA
```

- Funkcja `splice` jest bardziej uniwersalna od poprzednich
- Potrafi usuwać i dodawać elementy w dowolnym miejscu
- Zwraca wszystkie usunięte elementy w kontekście listowym
- Zwraca ostatni usunięty element w kontekście skalarnym
- Liczba indeksów jest odpowiednio zmniejszana lub zwiększana

Wstawianie i usuwanie dowolnych elementów tablicy II

- `splice` może zastąpić każdą z poprzednich funkcji

```
1  push(@a,$x,$y)          splice(@a,@a,0,$x,$y)
   pop(@a)                 splice(@a,-1)
   shift(@a)               splice(@a,0,1)
   unshift(@a,$x,$y)       splice(@a,0,0,$x,$y)
5  $a[$i] = $y             splice(@a,$i,1,$y)
```

Wstawianie i usuwanie dowolnych elementów tablicy II

- splice może zastąpić każdą z poprzednich funkcji

```
push(@a,$x,$y)          splice(@a,@a,0,$x,$y)
pop(@a)                 splice(@a,-1)
shift(@a)               splice(@a,0,1)
unshift(@a,$x,$y)       splice(@a,0,0,$x,$y)
$a[$i] = $y             splice(@a,$i,1,$y)
```

- i nie tylko

```
splice(@a,3,3)          # usuń 3 elementy od 3
splice(@a,3,3,$x,$y)    # zastąp 3 elementy od 3
                        # elementami $x i $y
splice(@a,3)            # usuń wszystkie elementy od 3
splice(@a)              # usuń wszystkie elementy
splice(@a,3,0,$x)       # wstaw $x na 3, przesuń resztę
```

Kontekst listowy a zmienne skalarne

Jakie będą odpowiednio wartości zmiennych $test, $test2 oraz @tablica2 w poniższych przykładach i dlaczego?

```
@tablica = qw(aa ab ba bb);

$test = @tablica;
print "Test 1: $test\n";

($test) = @tablica;
print "Test 2: $test\n";

($test, $test2) = @tablica;
print "Test 3: $test, $test2\n";

(@tablica2, $test) = @tablica;
print "Test 4: $test\n";
```

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Czym rózni się zmienna lokalna od globalnej?

- Zmienna globalna jest dostępna w każdym miejscu naszego programu

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Czym rózni się zmienna lokalna od globalnej?

- Zmienna globalna jest dostępna w każdym miejscu naszego programu
- Zmienna lokalna jest dostępna tylko wewnątrz bloku, w którym została stworzona

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Czym rózni się zmienna lokalna od globalnej?

- Zmienna globalna jest dostępna w każdym miejscu naszego programu
- Zmienna lokalna jest dostępna tylko wewnątrz bloku, w którym została stworzona
- Zmienna globalna przestaje istnieć, gdy zakończy się program

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Czym rózni się zmienna lokalna od globalnej?

- Zmienna globalna jest dostępna w każdym miejscu naszego programu
- Zmienna lokalna jest dostępna tylko wewnątrz bloku, w którym została stworzona
- Zmienna globalna przestaje istnieć, gdy zakończy się program
- Zmienne lokalna przestaje istnieć, gdy opuszczamy blok, w którym zostałą stworzona

Zmienne lokalne i zmienne globalne I

- Jak dotąd korzystaliśmy tylko ze zmiennych globalnych
- Ogólnie uznaję sie to za zły styl programowania – Dlaczego?

Czym rózni się zmienna lokalna od globalnej?

- Zmienna globalna jest dostępna w każdym miejscu naszego programu
- Zmienna lokalna jest dostępna tylko wewnątrz bloku, w którym została stworzona
- Zmienna globalna przestaje istnieć, gdy zakończy się program
- Zmienne lokalna przestaje istnieć, gdy opuszczamy blok, w którym zostałą stworzona
- Uwaga: Zmienna $_ jest zmienną globalną !

Zmienne lokalne i zmienne globalne II

```
$globalna = 4;
print "Globalna: $globalna\n";

{
  my $lokalna = 5;
  print "Lokalna: $lokalna\n";
}

print "Globalna: $globalna\n";
print "Lokalna: $lokalna\n";
```

Zmienne lokalne i zmienne globalne II

```
$globalna = 4;
print "Globalna: $globalna\n";

{
  my $lokalna = 5;
  print "Lokalna: $lokalna\n";
}

print "Globalna: $globalna\n";
print "Lokalna: $lokalna\n";
```

- Zmienna lokalna jest tworzona wewnątrz bloku za pomocą słowa kluczowego `my`

Zmienne lokalne i zmienne globalne II

```
$globalna = 4;
print "Globalna: $globalna\n";

{
  my $lokalna = 5;
  print "Lokalna: $lokalna\n";
}

print "Globalna: $globalna\n";
print "Lokalna: $lokalna\n";
```

- Zmienna lokalna jest tworzona wewnątrz bloku za pomocą słowa kluczowego `my`
- Bez słowa kluczowego `my` stworzylibyśmy kolejną zmienną globalną

Zmienne lokalne i zmienne globalne III

```
my $x = 4;
print "Blok 1: x = $x\n";
{
   my $x = 5;
   my $y = 3;
   print "Blok 2: x = $x, y = $y\n";
   {
      my $x;
      print "Blok 3: x = $x, y = $y\n";
   }
}
print "Blok 1: x = $x, y = $y\n";
```

- Zmienna lokalna stworzona poza jakimkolwiek blokiem jest zmienną globalną

Zmienne lokalne i zmienne globalne III

```
1  my $x = 4;
   print "Blok 1: x = $x\n";
   {
     my $x = 5;
5    my $y = 3;
     print "Blok 2: x = $x, y = $y\n";
     {
       my $x;
       print "Blok 3: x = $x, y = $y\n";
10   }
   }
   print "Blok 1: x = $x, y = $y\n";
```

- Zmienna lokalna stworzona poza jakimkolwiek blokiem jest zmienną globalną
- `my` tworzy zupełnie nową zmienną, która *przykrywa* inne zmienne (z otaczających bloków) o tej samej nazwie

Funkcje w Perlu

- Nasze programy zaczynają się rozrastać
- Byłoby lepiej, gdybyśmy znali sposób na uporządkowanie naszego kodu

Funkcje w Perlu

- Nasze programy zaczynają się rozrastać
- Byłoby lepiej, gdybyśmy znali sposób na uporządkowanie naszego kodu
- Takim sposobem są funkcje

Funkcje w Perlu

- Nasze programy zaczynają się rozrastać
- Byłoby lepiej, gdybyśmy znali sposób na uporządkowanie naszego kodu
- Takim sposobem są funkcje
- Funkcje to małe podprogramy, z których możemy korzystać wiele razy w naszym programie

Funkcje - przykład programu bez funkcji

```
print "Podaj imie: ";
while(my $name = <STDIN>) {
  chomp($name);
  exit if($name eq '');
  print "Witaj, $name!\n";
  print "Podaj imie: ";
}
```

Funkcje - przykład programu bez funkcji

```
print "Podaj imie: ";
while(my $name = <STDIN>) {
  chomp($name);
  exit if($name eq '');
  print "Witaj, $name!\n";
  print "Podaj imie: ";
}
```

- Podobny program widzieliśmy na ostatnim wykładzie
- Co się stanie, gdy podamy "pustę" imię?
- Dlaczego pojawia się dwa razy ciąg "Podaj imię"?

Funkcje - pierwszy przykład z funkcjami

```
1  while(my $name = pobierz_imie()) {
     print "Witaj, $name!\n";
   }

5  sub pobierz_imie {
     print "Podaj imie: ";
     my $name = <STDIN>;
     chomp($name);
     return $name;
10 }
```

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)
- Wykonują się one dopiero, gdy jawnie użyjemy nazwy funkcji we właściwym programie

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)
- Wykonują się one dopiero, gdy jawnie użyjemy nazwy funkcji we właściwym programie
- Słowo kluczowe `sub` poprzedza nazwę funkcji, którą ustalamy sami; nazwy powinny kojarzyć się działaniem tworzonej funkcji

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)
- Wykonują się one dopiero, gdy jawnie użyjemy nazwy funkcji we właściwym programie
- Słowo kluczowe `sub` poprzedza nazwę funkcji, którą ustalamy sami; nazwy powinny kojarzyć się działaniem tworzonej funkcji
- Polecenie `return` wewnętrz funkcji, natychmiast kończy działanie funkcji

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)
- Wykonują się one dopiero, gdy jawnie użyjemy nazwy funkcji we właściwym programie
- Słowo kluczowe `sub` poprzedza nazwę funkcji, którą ustalamy sami; nazwy powinny kojarzyć się działaniem tworzonej funkcji
- Polecenie `return` wewnętrz funkcji, natychmiast kończy działanie funkcji
- Jeśli przy `return` zostało podane jakieś wyrażenie, to zostanie one przez funkcję *zwrócone*

Struktura funkcji

- Funkcje to bloki programu, które są zapisywane poza właściwym programem (za lub przed)
- Wykonują się one dopiero, gdy jawnie użyjemy nazwy funkcji we właściwym programie
- Słowo kluczowe `sub` poprzedza nazwę funkcji, którą ustalamy sami; nazwy powinny kojarzyć się działaniem tworzonej funkcji
- Polecenie `return` wewnętrz funkcji, natychmiast kończy działanie funkcji
- Jeśli przy `return` zostało podane jakieś wyrażenie, to zostanie one przez funkcję *zwrócone*
- Możemy zwracać wartości skalarne i tablicowe

Przykład - przekazywanie argumentów do funkcji

```
while(my $name = pobierz_imie()) {
  witaj($name);
}

sub pobierz_imie {
  print "Podaj imie: ";
  my $name = <STDIN>;
  chomp($name);
  return $name;
}

sub witaj {
  my $name = shift;
  print "Witaj, $name!";
}
```

Przekazywanie argumentów do funkcji

- Możemy również przekazywać zmienne i wartości do funkcji

Przekazywanie argumentów do funkcji

- Możemy również przekazywać zmienne i wartości do funkcji
- Odbywa się to za pomocą specjalnej tablicy `@_`
- Tablica `@_` jest tablicową wersją zmiennej domyślnej `$_`

Przekazywanie argumentów do funkcji

- Możemy również przekazywać zmienne i wartości do funkcji
- Odbywa się to za pomocą specjalnej tablicy `@_`
- Tablica `@_` jest tablicową wersją zmiennej domyślnej `$_`
- Wszystkie argumenty funkcji są zapisywane do tablicy `@_` w takiej kolejności w jakiej zostały podane przy wywołaniu funkcji

Przekazywanie argumentów do funkcji

- Możemy również przekazywać zmienne i wartości do funkcji
- Odbywa się to za pomocą specjalnej tablicy `@_`
- Tablica `@_` jest tablicową wersją zmiennej domyślnej `$_`
- Wszystkie argumenty funkcji są zapisywane do tablicy `@_` w takiej kolejności w jakiej zostały podane przy wywołaniu funkcji
- Wewnątrz funkcji powinniśmy (ale nie musimy) skopiować wartości z tablicy `@_` do odpowiednio nazwanych zmiennych lokalnych
- Do kopiowania mogą nam służyć dowolne metody poznane dzisiaj w pierwszej części wykładu. Poprzedni i następny sposób uznaje się za kanoniczne.

Przykład - wersja wieloargumentowa (kanoniczny)

```
1  while(my ($f_name,$s_name) = pobierz_nazwisko()) {
     witaj($f_name,$s_name);
   }

5  sub pobierz_nazwisko {
     print "Podaj imie: ";
     my $f_name = <STDIN>;
     print "Podaj nazwisko: ";
     my $s_name = <STDIN>;
10
     chomp($f_name,$s_name);
     return ($f_name,$s_name);
   }

15 sub witaj {
     my ($f_name,$s_name) = @_;
     print "Witaj, $f_name $s_name!";
   }
```

Przykład - wersja wieloargumentowa (niekanoniczny)

```
while(my @names = pobierz_nazwisko()) {
  witaj(@names);
}

sub pobierz_nazwisko {
  my @names;
  print "Podaj imie: ";
  $names[0] = <STDIN>;
  print "Podaj nazwisko: ";
  $names[1] = <STDIN>;

  chomp(@names);
  return @names;
}

sub witaj {
  print "Witaj, $_[0] $_[1]!";
}
```

Program zliczający znaki w pliku - bez funkcji

```
1  my $total_count;
   foreach my $filename (@ARGV) {
     open(IN, "<$filename") or die "Plik nie istnieje";
     my $count;
5    while(<IN>) {
       $count += length($_);
     }
     close(IN);
     print "Plik $filename zawiera $count znakow\n";
10   $total_count += $count;
   }
   print "W sumie było $total_count znakow\n";
```

Program zliczający znaki w pliku - z funkcjami

```
my $total_count;
foreach my $filename (@ARGV) {
  my $count = zlicz($filename);
  print "Plik $filename zawiera $count znakow\n";
  $total_count += $count;
}
print "W sumie było $total_count znakow\n";

sub zlicz {
  my $filename = shift;
  open(IN, "<$filename") or die "Plik nie istnieje";
  my $count;
  while(<IN>) {
    $count += length($_);
  }
  close(IN);
  return $count;
}
```