Referencje w Perlu

Narzędzia informatyczne w językoznawstwie

Marcin Junczys-Dowmunt

21 lutego 2008

1 Wprowadzenie do referencji

Referencje to tak naprawdę nazwy zmiennych. Możemy traktować zmienne jako przedmioty rzeczywiste, a referencje jako ich oznaczenia. Tak jak w świecie rzeczywistym, istnieje tylko jeden konkretny obiekt, niemniej może on mieć wiele nazw. Referencje to trzeci typ danych skalarnych (a nie złożonych) w Perlu obok zmiennych liczbowych i łańcuchów znakowych.

1.1 Referencje do podstawowych typów zmiennych

Referencje do dowolnych zmiennych tworzymy za pomocą \. Ponieważ referencje to dane skalarne, przechowujemy je w zmiennych skalarnych, tak jak liczby lub łańcuchy znakowe.

```
1 use strict;

  my @array = (1, 2, 3);
  my %hash = (jeden => 1, dwa => 2, trzy => 3);
5 my $scalar = "Taki sobie skalar";

  my $aref = \@array;
  my $href = \%hash;
  my $sref = \$scalar;
```

1.2 Anonimowe listy i hasze

Powyżej zdefiniowaliśmy konkretne zmienne skalarne, tablicowe i haszowe, a następnie stworzyliśmy referencje do nich. W przypadku tablic i haszów można się obejść bez tego kroku pośredniego, tworząc anonimowe tablice i hasze.

```
1 use strict;

  my $aref = [ 1, 2, 3 ];
  my $href = { jeden => 1, dwa => 2, trzy => 3 };
```

Operator [...] tworzy listę (może być pusta) i zwraca automatycznie referencję do tej tablicy. Podobnie {...} tworzy hasz (także może być pusty) i zwraca odpowiednią referencję. Wewnątrz tych operatorów zachodzą odpowiednio konteksty listowe i haszowe.

1.3 Sposoby dostępu

```
use strict;

my @array = (1, 2, 3);
my %hash = (jeden => 1, dwa => 2, trzy => 3);
my $scalar = "Taki sobie skalar";

my $aref = \@array;
my $href = \%hash;
my $sref = \$scalar;

print "Tablica referowana przez \$aref ma ";
print  scalar @{$aref} . " elementow\n\n";

print "Hash referowany przez \$href ma ";
print scalar keys %{$href} . " kluczy\n\n";

print "Kanoniczny sposob dostepu: \n";
print "${$aref}[0] == $array[0]\n";
print "${$href}{jeden} == $hash{jeden}\n";
print "${$ref} == $scalar"

print "Skrocony sposob dostepu, gdy nie ma wieloznacznosci: \n";
print "$$aref[0] == $array[0]\n";
print "$$href{jeden} == $hash{jeden}\n";
print "$$ref == $scalar"

print "Notacja 'strzalkowa': \n";
print "$aref->[0] == $array[0]\n";
print "$href->{jeden} == $hash{jeden}\n";
```

1.4 Wygodne wyświetlanie

Funkcja Data::Dumper pozwala na wygodne i przejrzyste wyświetlanie złożonych struktur danych. Radzi sobie nawet z cyklicznymi strukturami.

```
use strict;
use Data::Dumper;

my $test = {
  liczbowo => [1, 2, 3],
  slownie => ['jeden', 'dwa', 'trzy'],
  cykl => $test,
};

print Dumper($test);
```

1.5 Uwaga!

```
use strict;
use Data::Dumper;

my @array = (1, 2, 3);

my $aref1 = \@array;  # referencja do @array
my $aref2 = [ @array ]  # referencja do kopii @array

$aref1->[0] = 4;
$aref2->[1] = 5;

print Dumper(\@array);
```

2 Tablica tablic

W perlu nie ma prawdziwych tablic wielowymiarowych. Można uzyskać ten sam efekt budując tablice tablic, które nie różnią się pod względem funkcji i wydajności od prawdziwych tablic wielowymiarowych. W zależności od liczby zagnieżdżeń można w ten sposób otrzymąc tablice n-wymiarowe.

2.1 Przykład: Indeksowany tekst

```
my @tekst;
while(<>) {
  chomp;
  push(@tekst, [ split(/[\s.,;:\-?!)(]+/, $_) ]);
}

print "Wyrazem na pozycji (3,5) jest: " . wyraz_n_m(3,5,\@tekst) . "\n";
print "Wyrazem na pozycji (10,2) jest: " . wyraz_n_m(10,2,\@tekst) . "\n";

sub wyraz_n_m {
  my ($n, $m, $tekst) = @_;
  if($n-1 < @$tekst and $m-1 < @{$tekst->[$n-1]}) {
    return $tekst->[$n-1]->[$m-1];
  }
  return "Nie ma takiego wyrazu";
}
```

3 Tablica haszów

Np. reprezentacja grafów za pomocą takiej struktury jest bardzo wygodna i oszczędna. Np. gdy wierzchołki mają reprezentację liczbową, można je traktować jak indeksy tablicy. Gdy taki graf ma mało krawędzi nie opłaca się kodować wierzchołków połączonych za pomocą tablicy, ponieważ tablica ma tyle elementów ile wyności jej największy indeks (plus jeden). Hasz będzie miał tylko tyle elementów ile będzie krawędzi wychodzących.

Nie mogę akurat wymyśleć prostego przykładu. Korzystałem już nie raz z takiej reprezentacji, ale były to złożone problemy (np. zliczanie kookurencji wyrazów w korpusach równoległych).

4 Hasz tablic

Hasz tablic jest idealną strukturą danych, gdy jednemu kluczowi odpowiada wiele wartości. Możemy je wtedy umieścić w tablicy, a referencję do tablicy jako wartość w haszu.

4.1 Przykład: Lemmatyzator

```
use strict;

my $lexicon = {
  analityk => [ 'analityka_R', 'analityk_R' ],
  analityka => [ 'analityka_R', 'analityk_R' ],
  analitykach => [ 'analityka_R', 'analityk_R' ],
  analitykami => [ 'analityka_R', 'analityk_R' ],
  analityki => [ 'analityka_R' ],
  analitykiem => [ 'analityk_R' ],
  analityko => [ 'analityka_R' ],
  analitykom => [ 'analityka_R', 'analityk_R' ],
  analitykowi => [ 'analityk_R' ],
  analityku => [ 'analityk_R' ],
  analityk\ow => [ 'analityk_R' ],
  analityk\a => [ 'analityka_R' , 'analityk_R' ],
  analityk\e => [ 'analityka_R' ],
  # ... dalsze wpisy
}

while(<>) {
  chomp;

  print "Lematyzuje zdanie: $_\n";
  my @tokens = split(/[\s.,;:\-?!)(]+/, $_);
  foreach my $token (@tokens) {
    print "$token : " . join("|", @{$lexicon->{lc($token)}}) ."\n";
  }
}
```

Wyrazy tekstowe nie zawsze można jednoznacznie przyporządkować do jednego lematu. Gdy uwzględnimy synktretyzm i inne informacje jak np. przypadek, liczba itp., to mamy jeszcze więcej wieloznaczności, czyli dłuższe listy.

5 Hasz haszów

Hasze zawierające hasze to kolejna bardzo przydatna struktura danych. Jest ona wolniejsza od struktury wykorzystująca tablice, ale zawiera tylko dokladnie tyle elementów ile trzeba.

5.1 Przykład: Wyszukiwanie bigramów

```
czerwony krzyz
czerwony baron
klucz zapadkowy
klucz udarowy
klucz dekodujacy
program dekodujacy
program rozwoju
...
```

Tablica 1: Przykładowe kolokacje bigramowe

```
use strict;

my %bigramy = (
  czerwony => {
    krzyz => 1,
    baron => 1,
  },
  klucz => {
    zapadkowy => 1,
    udarowy => 1,
    dekodujacy => 1,
  },
  program => {
    dekodujacy => 1,
    rozwoju => 1,
  },
);

while(<>) {
  my @tokens = split(/[\s.,;:\-?!)(]+/, $_);
  for(my $i = 0; $i+1 < @tokens; $i++) {
    if($bigramy{$tokens[$i]}->{$tokens[$j]}) {
      print "Znaleziono bigram : $tokens[$i] $tokens[$i+1]";
    }
  }
}
```

Przykład można oczwiście rozszerzyć do kolokacji o dowolnej liczbie wyrazów. Wtedy trzeba jednak stosować np. rekurencję w celu przeglądania struktury danych. Jest to przy okazji przykład na prosty automat skończony lub proste drzewo o dowolnej liczbie rozgałęzień.

6 Inne struktury danych

Za pomocą referencji do tablic i haszów można tworzyć struktury danych o dowolnej liczbie zagnieżdżeń. Jedynie rozmiar dostępniej pamięci nas ogranicza. Możemy zbudować drzewa, automaty skończone lub inne rodzaje grafów, traktować hasze jako rekordy, tablice jako n-tki itp. Należy też pamiętać, że jedną referencję można wykorzystać więcej niż raz, tworząc w ten sposób strukturę, nie powielając dane.

```
my $rodzice = [ qw(Antek Berta) ];
my $bracia = {
  Juzek => {
    rodzice => $rodzice,
    wiek => 20,
  },
  Benek => {
    rodzice => $rodzcie,
    wiek => 25,
  },
};

print Dumper($bracia);
```

Takie struktury przydają się przy tworzeniu automatów skończonych, za pomocą których można wydajnie zapamiętać i wyszukiwać np. dowolne kolokacje o dowolnej długości.