Niektóre rozwiązania wzorcowe do zestawu Perl 4

Narzędzia informatyczne w językoznawstwie

Marcin Junczys-Dowmunt

22 lutego 2008

Rozwiązania dotyczą „wersji z plusem” z wyjątkiem zadania 4.3, gdzie chyba nie ma eleganckiego sposobu by rozwiązać zadanie zgodnie z poleceniem.

Zadanie 4.2

```
1 use strict;
2
3 my %double;
4 while(<>) {
5     chomp;
6     $double{$_}++;
7 }
8 print "$_ - $double{$_}\n" foreach(keys %double);
```

Jeśli sobie przypomnimy, że wartości nieistniejące są w Perlu inicjalizowane automatycznie (w kontekście numerycznym zerem), to krok w wierszu nr 6 staje się zupełnie jasny. Sposób działania reszty programu jest chyba oczywisty.

Zadanie 4.3

```
1 use strict;
2
3 my %double = map { chomp; $_ => 1 } <>;
4 print "$_\n" foreach(keys %double);
```

Funkcja `map` działa na listach, operator diamentowy znajduję się więc w kontekście listowym i tym samym informacje na wejściu zostaną skonsumowane w całości, każdy wiersz jako element listy. Funkcja `map` tworzy na podstawie tej listy nową listę, gdzie elementy o nieparzystym indeksie to wczytane wiersze, a elementy parzyste to same jedynki. Przypisanie wyniku funkcji `map` do hasza powoduje zlikwidowanie powtarzających się kluczy. [1]

[1] Jest to wyłączna własność haszy! Przypisanie do tablicy nie spowodowałoby usunięcia powtarzających się elementów.

Zadanie 4.4

Wersja 1

```
use strict;

chomp(my @lines = sort <>);
my $counter = 1;
foreach my $i (1 .. @lines-1) {
    if($lines[$i-1] ne $lines[$i]) {
        print "$lines[$i-1] - $counter\n";
        $counter = 1;
    }
    else {
        $counter++;
    }
}
print "$lines[@lines-1] - $counter\n";
```

Tutaj nie korzystamy z haszy tylko z faktu, że po posortowaniu, powatarzające się wiersze będą występować kolejno po sobie. Wystarczy więc śledzić, kiedy dwa kolejne wiersze są różne i wypisać wtedy liczbę wystąpień, która była zwiększana tylko wtedy, gdy dwa kolejne wiersze się nie różniły. Ostatni wiersz programu zapewnia, że nie zapomnimy o ostatnim wierszu w liście.

Wersja 2

```
use strict;

chomp(my @lines = sort <>);
while(@lines) {
    my $line = shift @lines;
    my $counter = 1;
    while(@lines and $line eq $lines[0]) {
        shift @lines;
        $counter++;
    }
    print "$line - $counter\n";
}
```

Ta wersja jest trochę bardziej elegancka, ale nie widać na pierwszy rzut oka, co się naprawdę dzieje. Po wczytaniu posortowanych wierszy do tablicy, usuwamy kolejne wiersze z początku tablicy. Główna pętla usuwa pierwszy element z tablicy, a wewnętrzna pętla usuwa wszystkie następujące po nim duplikaty tego wiersza, zliczając je. Wewnętrzna pętla przestaje działać, gdy następny wiersz juz nie jest duplikatem lub gdy zostały usunięte wszystkie wiersze. Główna pętla przestaje działać, gdy zostaną usunięte wszystkie wiersze z tablicy. Mimo tego, że program zawiera zagnieżdżoną pętlę, nie działa on dłużej niż pierwsza wersja.

Zadanie 4.5

Wersja 1

```
use strict;
my %hash;
while(<>) {
    chomp;
    if(not exists($hash{$_})) {
        print $_."\n";
        $hash{$_} = 1;
    }
}
```

Tutaj również korzystamy z hasza, ale w przeciwieństwie do poprzednich zadań wypisujemy wiersze w czasie wczytywania pliku. Pomysł jest taki, że wyświetlamy wiersz, gdy pojawi się po raz pierwszy i tylko wtedy. Wystarczy więc prześledzić, czy dany wiersz już kiedyś wystąpił (czyli, czy znajduje się w haszu), a jeśli nie, to go wyświetlimy.

Wersja 2

```
use strict;
my %hash;
while(<>) {
    chomp;
    if(not exists($hash{$_})) {
        $hash{$_} = $.;
    }
}
print $_."\n" foreach(sort { $hash{$a} <=> $hash{$b} } keys %hash);
```

Innym poprawnym rozwiązaniem jest powyższy program. Tutaj najpierw wczytujemy cały plik do hasza, ale jako wartości hasza zapisujemy informacje o pierwszym jego wystąpieniu w pliku (i tylko o pierwszym) w postaci numeru wiersza. Na końcu wystarczy posortować klucze hasza (wiersze) według przypisanych im wartości (numery wierszy) i wyświetlić klucze według tego porządku.