Perl - pozostałe materiały

Tablica mieszająca - hash table
...stanowi tablicę, której indeksem nie musi być numer i nie musi być to kolejny numer. Indeksem może być dowolne wyrażenie. Dzięki temu możemy tworzyć szybkie tabele, w których odnajdujemy pojedyncze wartości np. do tłumaczenia jednego rodzaju tekstów na inne. Nazwę tablicy mieszającej poprzedzamy znaczkiem **%**. Czyli:

```
use warnings;
use strict;

my $k=$ENV{"LANG"};
print "System przedstawia swój język jako $k\n";
$k=substr($k,0,index($k,'.'));

my %LANGUAGES = ( en_US => "Angielski",
                  en_GB => "Angielski",
                  pl_PL => "Polski",
                  de_DE => "Niemiecki",
                  de_AT => "Niemiecki",
                  es_ES => "Hiszpański",
                  ru_RU => "Rosyjski",
                  uk_US => "Ukraiński",
                  cs_CZ => "Czeski",
                  sk_SK => "Słowacki",
                  C => "Domyślny programu"
                );

print "Język komunikatów w systemie to $LANGUAGES{$k}\n";
```

W tym programie używamy dwóch tablic haszujących. Pierwsza to %ENV - występuje ona zawsze w skrypcie i zawiera stan zmiennych środowiskowych, które widzimy po wydaniu polecenia **env**, zaś druga to tablica %LANGUAGES, którą zdefiniowaliśmy i wypełniliśmy w kodzie.
Wynik działania programu:

```
mcbx@m4800:/tmp$ ./a.pl
System przedstawia swój język jako pl_PL.UTF-8
Język komunikatów w systemie to Polski
mcbx@m4800:/tmp$
```

Tak więc **poszczególne elementy adresujemy konstrukcją $tablica{klucz}**.

Aby dodać element, należy po prostu taki element zaadresować i przypisać:

```
$LANGUAGES{”lt_LT”}=”Litewski”;
```

Podobnie możemy zmieniać wartości tablicy.
Aby usunąć element, używamy funkcji delete:

```
delete $LANGUAGES{”C”};
```

Iterowanie po elementach tablicy jest możliwe. Ze względu na fakt, ze zawsze można „zajrzeć” do tego co akurat znajduje się w konkretnym elemencie tablicy, możemy iterować **po kluczach** pętli:

```
foreach my $key (keys %LANGUAGES) {
      print ”$LANGUAGES{$key}\n”;
}
```

Makro qw
...czyli „Quote word” - użyteczne, gdy nie mamy ochoty „oprawiać” każdego elementu w cudzysłów, a elementy są rozdzielone spacjami:

```
mcbx@m4800:/tmp$ cat b.pl
#!/usr/bin/perl
use warnings;
use strict;

my @k=qw"Jeden Dwa Trzy cztery";
foreach (@k) {
        print "$_\n";
}
mcbx@m4800:/tmp$ ./b.pl
Jeden
Dwa
Trzy
cztery
```

Ze względu na to, że qw nie jest stricte funkcją biblioteczną, to zamiast znaczka ” możemy użyć znaków /, {}, [], ‘, !, @ i kilku jeszcze - zawsze wynikiem będzie „pocięty” łańcuch tekstu.

Traktowanie danych binarnych

O ile Perl zazwyczaj jest używany do działania z danymi tekstowymi, nic nie szkodzi na przeszkodzie by działać na plikach binarnych. Niezbędne jest jednak tłumaczenie z gotowych wartości zmiennych, jakie przetwarzamy w języku Perl na reprezentację binarną w danym typie danych - np. int, unsigned int, short, float, double - każdy typ inaczej jest zapisany w pamięci.

Odczyt zaczynamy od otwarcia pliku binarnego:

```
open (my $FH,'<',$ARGV[0]) or die ("Cannot open input file");
binmode $FH;
```

Polecenie **binmode** wyłącza wszelkie udogodnienia, które działają gdy pracujemy z plikami tekstowymi. O ile w Uniksach pominięcie binmode katastrofy nie spowoduje, o tyle **w Windowsach jest bardzo ważne**, bowiem zapisując plik binarny bez binmode, szczególnie przez printa, może okazać się, że ze względu na niezgodność znaków nowej linii wszystkie bajty o wartości 0x0A (nowa linia w Uniksie, \n) zostaną zamienione na 0x0D 0x0A - czyli nowa linia w Windowsie (\r\n), co przy plikach innych niż tekstowe spowoduje, że w innych programach pożytku z tego pliku nie będzie.

Odczytajmy cztery bajty z pliku binarnego i pozyskajmy z nich informację, zakładając, że jest to dwubajtowy unsigned short:

```
my $buf;
read ($FH, $buf, 2);
$value=unpack("v",$buf);
```

Pliki binarne posiadają zazwyczaj jakiś nagłówek (magic bytes, jeżeli opracowujemy własny sposób zapisu to w dobrym tonie jest dodawanie jakiegoś) a dane następują później. Jak przemieścić się do kolejnych bajtów?

```
seek($FH,255,0);
```

Następny odczyt będzie już od 256 bajtu w pliku.

Ćwiczenie:
Pod jaką architekturę jest skompilowany plik /usr/bin/perl? Informacja o architekturze znajduje się pod 0x12 od początku pliku wykonywalnego typu ELF.

Rozwiązanie:

```
16%{sendzimir}/home/stud/dokt/mwilkus>cat egzek.pl
#!/usr/local/bin/perl

use warnings;
use strict;

my %ARCH = ( 2 => "Sun SPARC", 3 => "Intel 32-bit", 0x3E => "Intel 64-bit");

open (my $INF,'<',$ARGV[0]) or die("Cannot open file");
binmode($INF);
seek($INF,0x11,0);
my $buf;
read($INF,$buf,2);
my $value=unpack("v",$buf);
print "Architektura to: $ARCH{$value}\n";

close($INF);
17%{sendzimir}/home/stud/dokt/mwilkus>./egzek.pl /usr/bin/perl
Architektura to: Sun SPARC
18%{sendzimir}/home/stud/dokt/mwilkus>
```

Ten samo kod na Intelu zaś:

```
mcbx@m4800:/tmp$ perl c.pl /usr/bin/perl
Architektura to: Intel 64-bit
mcbx@m4800:/tmp$
```

Zapis do pliku binarnego może zostać zrealizowany zarówno zapisywaniem bufora przez **write** analogicznie do read, jak i printem - ze względu na to, że zazwyczaj musimy zapisać wartości w pewien sposób (typ danych) analogicznie do unpack używamy funkcji **pack** i dopiero zapakowaną w odpowiedni typ wartość zapisujemy.

Drobna uwaga: W wersji intelowej kodu należało zmienić endianness (kolejność bajtów). Decyduje o tym wartość bajtu spod 0x05 - w Solarisie jest inna niż w Intelu!

Biblioteki
Perl posiada znaczną ilość bibliotek zarówno rozszerzających język, jak i będących pomostami do bibliotek w innych językach programowania. Użytkownicy Pythona oraz Tkinter zauważą, że istnieje biblioteka Tk zapewniająca graficzny interfejs użytkownika. Przykładowe okienko pokazano poniżej:

```
#!/usr/bin/perl

use warnings;
use strict;

use Tk 804;

my $mw = MainWindow->new;
my $lHandle= $mw->Label(-text => "Hello, World")->grid(-column=>0, -row=>0);
my $bHandle = $mw->Button(-text=>"Exit", -command=>sub {exit})->grid(-column=>0, -row=>1);

MainLoop;
```