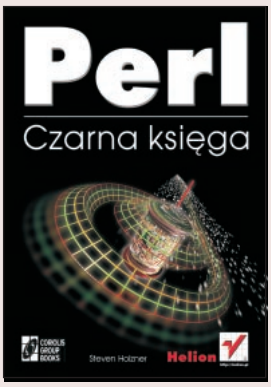
Perl. Czarna księga



Autor: Steven Holzner
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7197-288-1
Tytuł oryginału: Perl Core Language Little Black Book
Format: B5, stron: 416

Książka zawiera rzeczowe wprowadzenie do programowania w Perlu. Jeśli chcesz dowiedzieć się, jak coś zrobić w Perlu, tutaj znajdziesz odpowiedź.
Perl jest dużym i złożonym produktem, a niniejsza książka czyni ten język przystępnym. Jest to książka wyjątkowa, gdyż można ją uznać za bardzo zwarty podręcznik Perla 5. Autor to doskonały programista, który w swojej książce umieścił blisko 400 działających przykładów, a także omówił około 500 różnych zagadnień. Znajdziesz tu kompletny opis składni Perla 5, jak również opis tworzenia internetowych książek odwiedzin. Nauczysz się tworzyć perlowe klasy i obiekty, używać pakietów i modułów, łączyć Perla z pakietem Tk, czy choćby budować aplikacje do prowadzenia rozmów (chat). W książce tej są omówione praktyczne zagadnienia programowania w Perlu: obsługa cookies, kwestie bezpieczeństwa w Sieci, usuwanie błędów, wysyłanie ze skryptów CGI poczty elektronicznej, złożone struktury danych – znajdziesz tu po prostu wszystko.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 14.

Struktury danych

W skrócie

Najważniejszym mankamentem Perla, występującym aż do wersji 5, jest brak obsługi złożonych struktur danych, nawet wielowymiarowych tablic. Chyba właśnie tego programistom brakowało najbardziej — możliwości użycia w tablicy więcej niż jednego indeksu. Dotąd tablice trzeba było modelować na sposoby w najlepszym razie pokrętne. Konieczne było traktowanie indeksów tablic jako tekstu i łączenie ich w dłuższe zdania, które były używane jako klucze asocjacji — jak w poniższym przykładzie, gdzie tworzymy „tablicę” dwuwymiarową:

```
for $outerloop (0..4) {
   for $innerloop (0..4) {
      $array{"$outerloop, $innerloop"} = 1;
   }
}
```

Po utworzeniu takiej struktury danych można sięgać do elementów tej „tablicy”, podając indeksy złączone w jeden ciąg znaków:

```
print $array{'0, 0'};

1
```

Obecnie w Perlu dodano szeroką obsługę struktur danych, w tym tablice wielowymiarowe, można więc utworzyć następujący kod:

```
for $outerloop (0..4) {
   for $innerloop (0..4) {
      $array[$outerloop][$innerloop] = 1;
   }
}
print $array[0][0];

1
```

Jednak rzecz jest nieco bardziej skomplikowana, niż to się wydaje. Tablice i asocjacje Perla w zasadzie są jednowymiarowe, zatem w powyższym przykładzie utworzyliśmy tablicę wskaźników na inne tablice jednowymiarowe. Dzięki temu, że istnieje możliwość pominięcia operatorów dereferencji `->` między nawiasami, można pisać taki kod, jak powyżej, czyli na przykład `$array[$outerloop][innerloop]`, co jest równoważne `$array[$outerloop]->[innerloop]`. Trzeba jednak pamiętać, że tak naprawdę mamy tu do czynienia z tablicą wskaźników do tablic. Jeśli na przykład uruchomiona zostanie instrukcja `print @array`, nie pojawią się wcale elementy tablicy dwuwymiarowej, ale tylko wskaźniki do innych tablic jednowymiarowych:

```
ARRAY(0x8a56e4) ARRAY(0x8a578c)
ARRAY(0x8a58d0) ARRAY(0x8a5924)
ARRAY(0x8a5978)
```

Jako że tablica dwuwymiarowa tak naprawdę jest jednowymiarową tablicą wskaźników na inne tablice jednowymiarowe, można użyć konstruktora tablic anonimowych, `[]`:

```
$array[0] = ["jabłka", "pomarańcze"];
$array[1] = ["asparagus", "kukurydza", "groch"];
$array[2] = ["szynka", "kurczak"];
print $array[1][1];

kukurydza
```

To samo można zrobić również inaczej — w ten sposób inicjalizujemy `@array` listą wskaźników na tablice:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
print $array[1][1];

kukurydza
```

Tę tablicę stworzyliśmy, przekazując listę wskaźników na tablice. Jeśli zamiast nawiasów zwykłych zastosujemy nawiasy kwadratowe, w `$array` zostanie przekazany wskaźnik na anonimową tablicę tablic, do którego można później użyć operatora dereferencji `->`:

```
$array = [
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
];
print $array->[1][1];

kukurydza
```

Oprócz zapisywania w tablicy wskaźników na inne tablice, czasem można zobaczyć następujący kod, który tworzy tablicę dwuwymiarową:

```
@{$array[0]} = ("jabłka", "pomarańcze");
@{$array[1]} = ("asparagus", "kukurydza", "groch");
@{$array[2]} = ("szynka", "kurczak");
print $array[1][1];

kukurydza
```

Interpretacja zapisu `@{$array[0]}` jest dość złożona: Perl „wie", że konstrukcja `@{}` umożliwia dereferencję wskaźników na tablice, ale jako że `$array[0]` nie istnieje, Perl automatycznie taki element tworzy i zapisuje tam wskaźnik do tablicy zawierającej elementy z przypisanej listy. Ten sam proces powtarza się w odniesieniu do `$array[1]` i `$array[2]` — i tak oto powstaje dwuwymiarowa tablica. Jednak w przypadku stosowania takiego kodu trzeba zachować ostrożność, bo gdyby `$array[0]` już wcześniej istniała, to niezależnie od tego, na co by wskazywała, zostałaby nadpisana.

Po utworzeniu tablicy dwuwymiarowej można sięgać do jej elementów za pomocą indeksów:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
for $outer (0..$#array) {
   for $inner (0..$#{$array[$outer]}) {
      print $array[$outer][$inner], " ";
   }
   print "\n";
}
```

```
jabłka pomarańcze
asparagus kukurydza groch
szynka kurczak
```

Nadal istnieją techniki obsługi tablic jednowymiarowych, które można wykorzystać do obsługi tablic wielowymiarowych. W następnym przykładzie użyjemy indeksu pętli i metody `join` do wyświetlenia kolejnych tablic jednowymiarowych (zwróć uwagę na zastosowanie zapisu `@{}` do dereferencji wskaźnika na tablicę):

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
for $loopindex (0..$#array) {
   print join (", ", @{$array[$loopindex]}), "\n";
}
```

```
jabłka pomarańcze
asparagus kukurydza groch
szynka kurczak
```

Nie trzeba tu oczywiście stosować indeksu pętli, można użyć bezpośrednio tablicy wskaźników:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
for $arrayreference (@array) {
   print join (", ", @{$arrayreference}), "\n";
}
```

```
jabłka pomarańcze
```

```
asparagus kukurydza groch
szynka kurczak
```

Należy pamiętać, że naprawdę mamy do czynienia z tablicami tablic i to właśnie stanowi klucz do wszystkich struktur danych omawianych w tym rozdziale. Nie wolno zapominać, że podstawą są wskaźniki, i że nie są to typy podstawowe Perla.

Żądanie deklarowania zmiennych? Świetny pomysł!

Niejednokrotnie tworzenie struktur danych i obsługa wskaźników są złożonym zadaniem, w którego realizacji może pomóc użycie dyrektywy `use strict vars`. Załóżmy na przykład, że zapisano kod, ale zamiast `()` użyto `[]`, przypisując wskaźnik na tablicę anonimową `$array`, zamiast tablicy tablic:

```
use strict vars;
$array = [
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
];
print $array[0][0];
```

Interpreter Perla wygeneruje w ostatnim wierszu błąd, gdyż niejawnie używamy niezadeklarowanej zmiennej `@array`. Błąd ten wynika stąd, że zamiast zewnętrznych nawiasów kwadratowych powinny być użyte zwykłe nawiasy okrągłe, ewentualnie należałoby zastosować `$array` jako wskaźnik:

```
$array->[0][0]
```

Można tworzyć nie tylko tablice tablic i asocjacje asocjacji, ale również tworzyć konstrukcje mieszane:

```
$array[1][2]                    # tablica tablic
$hash{bigkey}{littlekey}        # asocjacja asocjacji
$array[3]{key}                  # tablica asocjacji
$hash{key}[4]                   # asocjacja tablic
```

Wszystkim tym typom przyjrzymy się dokładniej w dalszej części tego rozdziału.

Zagadnienia

Złożone rekordy: zapisywanie wskaźników i innych elementów

W strukturach danych Perla można przechowywać różnorodne dane, w tym wskaźniki na inne struktury, dzięki czemu można tworzyć złożone struktury połączone wskaźnikami. Tworzenie takich struktur może być bardzo przydatne do tworzenia kopii danych z różnych miejsc, aby zapewnić automatyczną aktualizację danych pierwotnych.

Najpierw jako przykład pokażemy zapisywanie różnych typów danych w strukturze, w tym wskaźników — także do procedur. Do zapisywania kopii tablic i asocjacji można użyć odpowiednich tablic i asocjacji anonimowych, ale można też zapisać wskaźniki na już istniejące tablice i asocjacje, co umożliwia użycie gotowych danych:

```
string = "Tutaj mamy jakiś napis.";
@array = (1, 2, 3);
%hash = ('owoc' => 'jabłka', 'warzywo' => 'kukurydza');
sub printem
{
   print shift;
}

$complex = {
   string         => $string,
   number         => 3.1415926,
   array          =>[@array],
   hash           => {%hash},
   arrayreference => \@array,
   hashreference  => \%hash,
   sub            => \&printem,
   anonymoussub   => sub {print shift;},
   handle         => \*STDOUT,
};
print $complex->{string}, "\n";
print $complex->{number}, "\n";
print $complex->{array}[0], "\n";
print $complex->{hash}{owoc}, "\n";
print ${$complex->{arrayreference}}[0], "\n";
print ${$complex->{hashreference}}{"owoc"}, "\n";
$complex->{sub}->("Wywołanie procedury.\n");
$complex->{anonymoussub}->("Wywołanie procedury anonimowej.\n");
print {$complex->{handle}} "Tekst drukowany do uchwytu pliku.\n";

Tutaj mamy jakiś napis.
3.1415926
1
jabłka
1
jabłka
Wywołanie procedury.
Wywołanie procedury anonimowej.
Tekst drukowany do uchwytu pliku.
```

Deklarowanie tablic tablic

Stosując tablice tablic (a nawet tablice tablic tablic i tak dalej), można tworzyć tablice wielowymiarowe, które są bezcenne, kiedy trzeba poindeksować dane względem więcej niż jednego wskaźnika (na przykład tablica studentów ze wskaźnikami: numer studenta i numer egzaminu). Oto najczęściej stosowana metoda tworzenia tablic tablic. Zauważ, że do tworzenia tablicy jednowymiarowej używamy konstruktora tablicy anonimowej:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
```

Tworzenie tablic tablic na bieżąco

Do tworzenia tablic tablic kawałek po kawałku można użyć konstruktora tablic anonimowych, wypełniając tablicę wskaźnikami do innych tablic jednowymiarowych:

```
$array[0] = ["jabłka", "pomarańcze"];
$array[1] = ["asparagus", "kukurydza", "groch"];
$array[2] = ["szynka", "kurczak"];
print $array[1][1];

kukurydza
```

To samo można zrobić, wymuszając na Perlu automatyczne tworzenie wskaźników (zajrzyj na początek tego rozdziału):

```
${array[0]} = ("jabłka", "pomarańcze");
${array[1]} = ("asparagus", "kukurydza", "groch");
${array[2]} = ("szynka", "kurczak");
print $array[1][1];

kukurydza
```

Można oczywiście tworzyć i wypełniać tablicę tablic element po elemencie:

```
for $outerloop (0..4) {
   for $innerloop (0..4) {
      $array[$outerloop][$innerloop] = 1;
   }
}
print $array[0][0];

1
```

Istnieje też możliwość wykorzystania instrukcji `push` do umieszczania wskaźników w tablicy tablic:

```
for $loopindex (0..4) {
   push @array, [1, 1, 1, 1];
}
print $array[0][0];

1
```

Oto kolejny przykład, w którym używamy listy zwracanej przez procedurę i konstruktora tablic anonimowych:

```
for $loopindex (0..4) {
   $array[$loopindex] = [&zerolist];
}
sub zerolist
{
   return (0, 0, 0, 0);
}
print $array[1][1];

1
```

Można też zawsze dodać do tablicy tablic nowy wiersz:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
$array[3] = ["makaron z kurczakiem", "chili"];
print $array[3][0];

makaron z kurczakiem
```

Równie dobrze można za pomocą metody `push` wstawić elementy do już istniejącego wiersza:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
push @{$array[0]}, "banan";
print $array[0][2];

banan
```

Sięganie do tablic tablic

Do tablicy tablic można sięgać element po elemencie:

```
for $outerloop (0..4) {
   for $innerloop (0..4) {
      $array[$outerloop][$innerloop] = 1;
   }
}
print $array[0][0];

1
```

Warto też sobie rzecz uprościć i użyć — typowej dla Perla — metody sięgania do tablic jednowymiarowych, jak w poniższym przykładzie, znanym już z początku tego rozdziału. Instrukcję `join` wykorzystujemy do złączenia tekstu ze wszystkich wierszy tablicy w całość:

```
@array = (
    ["jabłka", "pomarańcze"],
    ["asparagus", "kukurydza", "groch"],
    ["szynka", "kurczak"],
);
for $arrayref (@array) {
   print join(", ", @{$arrayref}), "\n";
}

jabłka pomarańcze
asparagus kukurydza groch
szynka kurczak
```

Deklarowanie asocjacji asocjacji

Asocjacji asocjacji używa się, kiedy jest potrzebny wielopoziomowy tekstowy system informacyjny, na przykład system ekspercki. W takim przypadku używamy tekstu do zagłębiania się na kolejne poziomy struktury danych. Aby od razu utworzyć asocjację asocjacji, można użyć takiej oto deklaracji:

```
%hash = (
   owoce => {
      ulubione => "jabłka",
      'też dobre' => "pomarańcze",
   },
   warzywa => {
      ulubione => "kukurydza",
      'też dobre' => "groszek",
      'nielubiane' => "rzepa",
   },
   'mięsa' => {
      ulubione => "kurczak",
      'też dobre' => "wołowina",
   },
);
print $hash{owoce}{ulubione};

jabłka
```

Jako wartości w parach klucz + wartość przypisywaliśmy następne asocjacje.

Tworzenie asocjacji asocjacji na bieżąco

Aby utworzyć asocjację asocjacji krok po kroku, dodajemy kolejne asocjacje z innymi kluczami:

```
$hash{owoce} = {
   ulubione => "jabłka",
   'też dobre' => "pomarańcze",
};
$hash{warzywa} = {
   ulubione => "kukurydza",
   'też dobre' => "groszek",
   'nielubiane' => "rzepa",
};
$hash{'mięsa'} = {
   ulubione => "kurczak",
   'też dobre' => "wołowina",
};
print $hash{owoce}{ulubione};

jabłka
```

Oto sposób tworzenia asocjacji przy pomocy konstruktora asocjacji anonimowych `{}` oraz par klucz + wartość, zwracanych przez procedurę:

```
for $key ("hash1", "hash2", "hash3") {
   $hash{$key} = {&returnlist};
}
sub returnlist
```

```
{
   return (key1 => value1, key2 => value2);
}
print $hash{hash1}{key2};

value2
```

Sięganie do asocjacji asocjacji

Aby pobrać wartości z asocjacji asocjacji, trzeba się do nich jawnie odwołać:

```
%hash = (
   owoce => {
      ulubione => "jabłka",
      'też dobre' => "pomarańcze",
   },
   warzywa => {
      ulubione => "kukurydza",
      'też dobre' => "groszek",
      'nielubiane' => "rzepa",
   },
);
print $hash{owoce}{'też dobre'};

pomarańcze
```

Używając standardowych technik obsługi asocjacji, możemy zrealizować pętlę po wszystkich elementach asocjacji:

```
%hash = (
   owoce => {
      ulubione => "jabłka",
      'też dobre' => "pomarańcze",
   },
   warz => {
      ulubione => "kukurydza",
      'też dobre' => "groszek",
   },
);
for $food (keys %hash) {
   print "$food\t {";
   for $key (keys %{$hash{$food}}) {
      print "'$key' => '$hash{$food}{$key}'";
   }
   print "}\n";
}

warz    {'ulubione' => 'kukurydza''też dobre' => 'groszek'}
owoce   {'ulubione' => 'jabłka''też dobre' => 'pomarańcze'}
```

Do uporządkowania pierwszego poziomu asocjacji należy użyć następującego wyrażenia:

```
owoce   {'ulubione' => 'jabłka''też dobre' => 'pomarańcze'}
warz    {'ulubione' => 'kukurydza''też dobre' => 'groszek'}
```

Deklarowanie tablic asocjacji

Tablicę asocjacji stosujemy do indeksowania rekordów oznaczonych kluczami (przykład zobaczymy w ostatnim punkcie tego rozdziału, kiedy będziemy tworzyć bufor pierścieniowy). Oto sposób tworzenia tablicy asocjacji od razu w pojedynczej deklaracji:

```
@array = (
   {
      ulubione => "jabłka",
      'też dobre' => "pomarańcze",
   },
   {
      ulubione => "kukurydza",
      'też dobre' => "groszek",
      'nielubiane' => "rzepa",
   },
   {
      ulubione => "kurczak",
      'też dobre' => "wołowina",
   },
};
print $$array[0]{ulubione};

jabłka
```

Tworzenie tablic asocjacji na bieżąco

Tablice asocjacji można też tworzyć, stopniowo przypisując asocjacje elementom tablicy:

```
@array[0] = {ulubione => "jabłka",
             'też dobre' => "pomarańcze"};
@array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nielubiane' => "rzepa"};
@array[2] = {ulubione => "kurczak",
             'też dobre' => "wołowina"};
print $array[0]{ulubione};

jabłka
```

Tak samo jak w przypadku wszelkich innych tablic, można użyć metody `push`:

```
push @array {ulubione => "jabłka",
             'też dobre' => "pomarańcze"};
push @array {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nielubiane' => "rzepa"};
push @array {ulubione => "kurczak",
             'też dobre' => "wołowina"};
print $array[0]{ulubione};

jabłka
```

Oto przykład, w którym odczytujemy pary klucz+wartość i rozbijamy je na tablicę asocjacji:

```
$data[0] = "ulubione:jabłka,też dobre:pomarańcze";
$data[1] = "ulubione:kukurydza,też dobre:groszek,
            nielubiane:rzepa";
$data[2] = "ulubione:kurczak,też dobre:wołowina";
for $loopindex (0..$#data) {
```

```
   for $element (split ',', $data[$loopindex]) {
      ($key, $value) = split ':', $element,
      $array[$loopindex]{$key} = $value;
   }
}
print $array[0]{'też dobre'};

pomarańcze
```

Sięganie do tablic asocjacji

Do tablic asocjacji sięgamy, używając po prostu indeksu tablicy i klucza wskazanej tym indeksem asocjacji:

```
@array[0] = {ulubione => "jabłka",
             'też dobre' => "pomarańcze"};
@array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nielubiane' => "rzepa"};
@array[2] = {ulubione => "kurczak",
             'też dobre' => "wołowina"};
print $array[0]{ulubione};

jabłka
```

Oto przykład, w którym wyświetlamy wszystkie elementy tablicy asocjacji, tworząc pętlę po wszystkich elementach:

```
@array[0] = {ulubione => "jabłka",
             'też dobre' => "pomarańcze"};
@array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nielubiane' => "rzepa"};
@array[2] = {ulubione => "kurczak",
             'też dobre' => "wołowina"};
for $loopindex (0..$#array) {
   print "array[$loopindex]: {";
   for $key (keys %{$array[$loopindex]}) {
      print "'$key' => '$array[$loopindex]{$key}' ";
   }
   print "}\n";
}

array[0] = {'ulubione' => 'jabłka' 'też dobre' => 'pomarańcze' };
array[1] = {'ulubione' => 'kukurydza' 'też dobre' => 'groszek'
   'nielubiane' => 'rzepa' }
array[2] = {'ulubione' => 'kurczak' 'też dobre' => 'wołowina' }
```

Ten sam efekt możemy uzyskać, stosując wskaźniki zamiast indeksu pętli:

```
@array[0] = {ulubione => "jabłka",
             'też dobre' => "pomarańcze"};
@array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nielubiane' => "rzepa"};
@array[2] = {ulubione => "kurczak",
             'też dobre' => "wołowina"};
for $hashreference (@array) {
   print "{";
   for $key (sort keys %hashreference) {
      print "'$key' => '$hashreference->{$key}'";
   }
```

```
    print "}\n";
}

{'ulubione' => 'jabłka' 'też dobre' => 'pomarańcze' };
{'ulubione' => 'kukurydza' 'też dobre' => 'groszek'
 'nielubiane' => 'rzepa' }
{'ulubione' => 'kurczak' 'też dobre' => 'wołowina' }
```

Deklarowanie asocjacji tablic

Jeśli są potrzebne dane indeksowane, dostępne przez wartość kluczową, można zastosować asocjację tablic.

Spośród wszystkich czterech kombinacji łączenia tablic z asocjacjami właśnie asocjacje tablic są najrzadziej spotykane.

Poniższy przykład pokazuje deklarowanie asocjacji tablic w jednym kroku:

```
@hash = (
   owoce => ["jabłka", "pomarańcze"],
   warzywa => ["kukurydza", "groszek", "rzepa"],
   'mięso' => ["kurczak", "szynka"],
);
print $hash{owoce}[0];

jabłka
```

Tworzenie asocjacji tablic na bieżąco

Asocjacje tablic można tworzyć na bieżąco. Trzeba w tym celu zapisywać tablice, które odpowiadają kolejnym kluczom asocjacji, przy pomocy konstruktora tablic anonimowych:

```
$hash{owoce} = ["jabłka", "pomarańcze"];
$hash{warzywa} = ["kukurydza", "groszek", "rzepa"];
$hash{'mięso'} = ["kurczak", "szynka"];
print $hash{owoce}[0];

jabłka
```

Można też użyć instrukcji `push`:

```
push @{$hash{owoce}}, "jabłka", "pomarańcze";
push @{$hash{warzywa}}, "kukurydza", "groszek", "rzepa";
push @{$hash{'mięso'}}, "kurczak", "szynka";
print $hash{owoce}[0];

jabłka
```

Sięganie do asocjacji tablic

Do wskazanego elementu tablicy tablic z asocjacji można sięgnąć w następujący sposób:

```
@hash = (
   owoce => ["jabłka", "pomarańcze"],
   warzywa => ["kukurydza", "groszek", "rzepa"],
   'mięso' => ["kurczak", "szynka"],
);
print $hash{owoce}[0];

jabłka
```

Oto przykład, w którym wyświetlamy całą asocjację tablic, używając instrukcji `join` do przekształcenia tablicy na tekst:

```
@hash = (
   owoce => ["jabłka", "pomarańcze"],
   warzywa => ["kukurydza", "groszek", "rzepa"],
   'mięso' => ["kurczak", "szynka"],
);
for $key (sort keys %hash) {
   print "$key:\t[", join(", ", @{$hash{$key}}), "]\n"
}

owoce: [jabłka, pomarańcze]
mięso: [kurczak, szynka]
warzywa: [kukurydza, groszek, rzepa]
```

Listy powiązanych wartości i bufory pierścieniowe

Używając struktur danych opisanych w tym rozdziale, bez problemu można utworzyć typowe struktury, takie jak drzewa binarne — gdzie dane zapisuje się węzłach łączonych przez gałęzie — albo listy powiązane. Listy powiązane składają się z danych zapisywanych w elementach, które same też są listami. Każdy element wskazuje na element następny w liście (a w listach powiązanych podwójnie — także na element poprzedni), dzięki czemu można taką listę przeglądać element po elemencie.

Często stosowaną postacią listy powiązanej jest bufor pierścieniowy, który powstaje po połączeniu listy w pierścień. Aby używać danych z bufora pierścieniowego, stosuje się dwa indeksy elementów: `head` i `tail`. Przy zapisywaniu korzysta się z `tail`, przy odczycie z `head`. Kiedy `head` i `tail` pokrywają się, bufor jest pusty. Dane odczytuje się i zapisuje przez przesuwanie `head` i `tail`, dzięki czemu bufory pierścieniowe efektywnie wykorzystują dostępną pamięć (na przykład w komputerach IBM PC i ich klonach klawisze wciskane przez użytkownika są zapisywane właśnie w buforze pierścieniowym. Kiedy ich liczba przekroczy 15, komputer informuje za pomocą sygnał dźwiękowego o przepełnieniu bufora).

Poniżej znajduje się przykład, który pokazuje utworzenie — przy użyciu tablicy asocjacji — bufora pierścieniowego z czterema elementami (można zatem zapisać w nim trzy elementy; zapisanie czwartego spowodowałoby, że `head` i `tail` znalazłyby się w tym samym miejscu, który to stan jest nieodróżnialny od pustego bufora). Każdy element bufora (a więc i element tablicy) jest asocjacją z dwoma kluczami: data, który odpowiada danym elementu, oraz next, który jest indeksem tablicy do następnego elementu listy. Oto sposób utworzenia samego bufora i ustawienia `head` i `tail` na to samo miejsce, co interpretuje się jako pusty bufor:

```
$buffer[0]{next} = 1;
$buffer[0]{data} = 0;
$buffer[1]{next} = 2;
$buffer[1]{data} = 0;
$buffer[2]{next} = 3;
$buffer[2]{data} = 0;
$buffer[3]{next} = 0;
$buffer[3]{data} = 0;
$head = 0;
$tail = 0;
```

Jeśli jakieś dane mają być zapisane, trzeba sprawdzić, czy bufor nie jest już pełny — jeśli jest, zwraca się wartość `false`. W przeciwnej sytuacji nową wartość się dopisuje, przesuwa się `tail` w przód i zwraca się wartość `true`:

```
sub store
{
   if ($buffer[$tail]{next} != $head) {   # Czy bufor jest pełny?
      $buffer[$tail]{data} = shift;
      $tail = $buffer[$tail]{next};
      return 1;
   } else {
      return 0;
   }
}
```

Jeśli dane należy pobrać, sprawdza się najpierw, czy bufor nie jest pusty — jeśli jest, zwracana jest wartość `undef`. W przeciwnym razie zwracana jest wartość wskazywana przez `head` i `tail` jest przesuwana w przód:

```
sub retrieve
{
   if ($head != $tail) {  # $tail == $head => pusty bufor
      $data = $buffer[$head]{data};
      $head = $buffer[$head]{next};
      return $data;
   } else {
      return undef;
   }
}
```

Spójrzmy teraz przykład zapisywania wartości w buforze, a potem ich pobierania:

```
store 0;
store 1;
store 2;
store 3;      # bufor pełny – wartość niezapisana
print retrieve, "\n";
print retrieve, "\n";
print retrieve, "\n";
```

```
0
1
2
```

Mimo że próbowaliśmy zapisać cztery wartości, to po trzech bufor był już pełny i dlatego czwarta wartość została pominięta.