

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Perl. Czarna księga

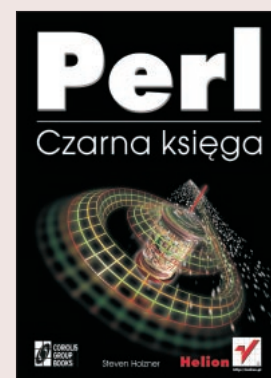
Autor: Steven Holzner

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

O Autorze	17
Wprowadzenie	19
Zawartość tej książki.....	19
Co będzie nam potrzebne?	22
Inne zasoby	23
Część I Składnia języka Perl.....	25
Rozdział 1. Co o Perlu trzeba wiedzieć?.....	27
W skrócie	27
Zaczynamy od początku	27
Zagadnienia	28
Uzyskanie i instalacja Perla	28
Zapisywanie skryptów Perla	29
Uwidacznianie Perla dla skryptów	30
Kod Perla: instrukcje i deklaracje	32
Uruchamianie skryptów Perla.....	33
Interaktywne uruchamianie skryptów Perla	35
Użycie przełączników wiersza poleceń	36
Użycie przełącznika -w — ostrzeżenia	38
Obsługa wejściowych i wyjściowych danych tekstowych przy pomocy standardowych uchwytów plików	39
Wyświetlanie danych tekstowych.....	39
Wyświetlanie numerów wierszy i nazw plików ze skryptami.....	40
Wielokrotne drukowanie tekstu	40
Podstawowe formatowanie tekstu	41
Wyświetlanie tekstu niesformatowanego: otwierane dokumenty Perla	42
Komentowanie kodu	42
Odczyt wprowadzanych danych	43
Użycie zmiennej domyślnej \$_	43
Czyszczenie danych wejściowych	44
Unikanie natychmiastowego zamknięcia okna skryptu w Windows 95/98 i NT	45
Rozdział 2. Zmienne skalarne i listy	47
W skrócie	47
Zmienne skalarne	47
Listy	48
Zagadnienia	48
Czym jest zmienna skalarna?.....	48
Nazwy zmiennych skalarnych	49

Przypisywanie wartości skalarnych	50
Co to jest lvalue?	50
Użycie liczb jako wartości skalarnych	50
Użycie wartości nieokreślonej: undef	51
Deklarowanie stałej	52
Obsługa w Perlu wartości logicznych	52
Przekształcanie liczb ósemkowych, dziesiętnych i szesnastkowych	52
Zaokrąglenie liczb	53
Użycie w zmiennych skalarnych tekstów	54
Stosowanie interpolacji w tekście	56
Skomplikowane interpolacje	57
Obsługa cudzysłowów i samodzielnych słów	58
Czym jest lista?	59
Odwolywanie się do elementów listy przez indeks	60
Przypisywanie listy innej liście	61
Odwzorowanie listy	61
Łączenie listy w ciąg znaków	61
Przekształcanie tekstu w listę	62
Sortowanie list	62
Odwracanie listy	63
Wybieranie elementów z listy	63
Jak rozumieć konteksty skalarny i listowy?	64
Wymuszanie kontekstu skalarnego	64
Rozdział 3. Tablice i asocjacje	67
W skrócie	67
Tablice	67
Asocjacje	67
Typy ogólne	68
Zagadnienia	68
Tworzenie tablic	68
Użycie tablic	70
Dokładanie elementów na koniec tablicy i ich usuwanie	70
Przesuwanie elementów tablicy	71
Określanie wielkości tablicy	72
Powiększanie i zmniejszanie tablic	72
Łączenie dwóch tablic	73
Pobieranie warstw tablic	73
Pętle na tablicach	73
Wyświetlanie tablic	75
Warstwy tablic	76
Odwracanie tablic	76
Sortowanie tablic	77
Tworzenie asocjacji	77
Użycie asocjacji	79
Dodawanie elementów do asocjacji	79
Sprawdzanie, czy istnieje dany element asocjacji	80
Usuwanie elementów asocjacji	80
Przetwarzanie asocjacji w pętli	81
Wyświetlanie asocjacji	82
Sortowanie asocjacji	82
Łączenie dwóch asocjacji	83
Użycie asocjacji i tablic w przypisaniach list	83

Użycie typów ogólnych	84
Porównywanie typów ogólnych i haseł tablicy symboli	85
Rozdział 4. Operatory i priorytety	87
W skrócie	87
Priorytety operatorów	88
Zagadnienia	89
Najwyższy priorytet: termy i lewostronne operatory list	89
Operator „strzałka”	90
Inkrementacja i dekrementacja	90
Potęgowanie	91
Symboliczne operatory unarne	91
Operatory wiązania	92
Mnożenie i dzielenie	92
Dodawanie, odejmowanie i konkatencja	93
Operatory przesunięć	93
Nazwane operatory unarne	94
Użycie operatorów testowania plików	94
Operatory porównania	95
Użycie operatorów równości	97
Iloczyn bitowy	98
Bitowa alternatywa	98
Bitowa alternatywa wyłączająca	98
Użycie logicznej koniunkcji w stylu C	99
Użycie logicznej alternatywy w stylu C	100
Operator zakresu	100
Operator wyboru	101
Przypisania	102
Przecinek jako operator	102
Użycie prawostronnych operatorów list	103
Logiczna negacja	103
Logiczna koniunkcja	103
Logiczna alternatywa	103
Logiczna alternatywa wykluczająca	104
Rozdział 5. Instrukcje warunkowe i pętle	105
W skrócie	105
Instrukcje warunkowe	105
Pętle	106
Zagadnienia	107
Instrukcja if	107
Instrukcja unless	108
Pętla for	109
Pętla foreach	110
Pętla while	112
Pętla until	113
Użycie if, unless, until i while do modyfikowania działania instrukcji	113
Pętla do while	114
next — wymuszenie następnej iteracji	114
last — zakończenie pętli	115
redo — powtarzanie iteracji	116
Instrukcja switch	116
Użycie goto	117
Użycie eval do uruchamiania kodu	118

Użycie exit do zakończenia wykonywania programu	118
Instrukcja die	119
Rozdział 6. Wyrażenia regularne.....	121
W skrócie	121
Użycie wyrażeń regularnych	121
Zagadnienia	123
Wyrażenia regularne	123
Zwykłe znaki w wyrażeniach regularnych	124
Klasy znaków w wyrażeniach regularnych	126
Alternatywne wzorce	127
Kwantyfikatory w wyrażeniach regularnych	127
Asercje w wyrażeniach regularnych	128
Odwołania do poprzednich dopasowań	129
Rozszerzenia wyrażeń regularnych	130
Modyfikatory operatorów m// i s//	130
Operator tr// — translacja	131
Użycie modyfikatorów operatora tr//	132
Dopasowywanie słów	132
Dobieranie początku wiersza	133
Dobieranie końca wiersza	133
Badanie liczb	133
Badanie liter	134
Odnajdowanie wielokrotnych dopasowań	134
Pomijanie wielkości liter	135
Wycinanie części ciągu	135
Wywołania funkcji i wyrażeń Perla w wyrażeniach regularnych	136
Dobranie n-tego dopasowania	136
Ograniczanie apetytu kwantyfikatorów	136
Usuwanie spacji wiodących i końcowych	137
Użycie asercji do sprawdzania w przód i wstecz	137
Rozdział 7. Procedury	139
W skrócie	139
Użycie procedur	139
Zagadnienia	140
Deklarowanie procedur	140
Prototypy	141
Definiowanie procedur	142
Wywoływanie procedur	143
Odczytywanie parametrów procedury	143
Zmienna liczba parametrów	144
Użycie domyślnych wartości parametrów	144
Wartości zwracane przez procedury (funkcje)	145
Użycie my do określania zasięgu	146
Żądanie leksykalnego ograniczenia zmiennych	148
Local — zmienne tymczasowe	148
Zmienne trwałe (statyczne)	149
Rekurencyjne wywoływanie procedur	150
Zagnieżdżanie procedur	150
Przekazywanie przez referencję	151
Przekazywanie pozycji z tablicy symboli	152
Użycie wantarray do sprawdzenia kontekstu wyjściowego	153
Funkcje wywoływane	153

Nadpisywanie funkcji wbudowanych i użycie CORE.....	154
Tworzenie procedur anonimowych	154
Tworzenie tablic procedur	155
Rozdział 8. Wskaźniki w Perlu.....	157
W skrócie	157
Wskaźniki bezpośrednie	157
Wskaźniki symboliczne	158
Operator strzałki	159
Anonimowe tablice, asocjacje i procedury	159
Zagadnienia	159
Tworzenie wskaźnika	159
Tworzenie wskaźników na tablice anonimowe	161
Tworzenie wskaźników na asocjacje anonimowe	162
Tworzenie wskaźników na procedury anonimowe.....	162
Tworzenie wskaźników na podstawie tablicy symboli	163
Dereferencja wskaźnika.....	164
Dereferencja przy użyciu operatora „strzałka”.....	165
Pomijanie operatora „strzałki”.....	166
Użycie operatora ref do określania typu wskaźnika	167
Tworzenie wskaźników symbolicznych.....	167
Wyłączanie wskaźników symbolicznych	168
Użycie wskaźników na tablice jako wskaźników na asocjacje	169
Tworzenie trwałych obszarów zasięgu	169
Tworzenie funkcji na podstawie szablonów	170
Część II Elementy wbudowane	173
Rozdział 9. Wbudowane zmienne Perla	175
W skrócie	175
Angielskie wersje wbudowanych zmiennych.....	175
Wiązanie zmiennych wbudowanych z uchwytami plików	177
Zagadnienia	178
\$' — tekst za wzorcem.....	178
\$- — numery wierszy w lewej części strony	178
\$! — ostatni błąd	179
\$" — separator pól wyjściowych interpolowanych wartości z tablic.....	179
\$# — format wyjściowy liczb.....	179
\$\$ — numer procesu Perla	179
\$\$ — numer strony wyjściowej.....	180
\$& — ostatnie dopasowanie	180
\$(— rzeczywisty GID.....	180
\$) — obowiązujący GID.....	180
\$* — wzorce wielowierszowe	180
\$, — separator pól wyjściowych	181
\$. — numer wiersza wejściowego	181
\$/ — separator rekordów wejściowych	181
\$: — znaki dzielenia wiersza	181
\$; — separator indeksów	182
\$? — status ostatniego zamknięcia potoku, wywołania polecenia przez `	
lub wywołania funkcji systemowej	182
\$@ — błąd ostatniej funkcji eval	182
\$[— początek indeksów tablic.....	182
\$\ — separator rekordów wyjściowych	183
\$] — wersja Perla	183

\$^	— format nagłówka strony	183
\$^A	— zapis bufora	183
\$^D	— flagi debugowania	183
\$^E	— informacje o błędzie związane z systemem operacyjnym	183
\$^F	— największy systemowy deskryptor pliku	184
\$^H	— ustawienia kontroli składniowej	184
\$^I	— wartość edycji w miejscu	184
\$^L	— zmiana strony wyjściowej	184
\$^M	— awaryjny bufor pamięci	184
\$^O	— nazwa systemu operacyjnego	184
\$^P	— obsługa debugowania	185
\$^R	— wynik asercji ostatniego wyrażenia regularnego	185
\$^S	— stan interpretera	185
\$^T	— czas uruchomienia skryptu	185
\$^W	— ustawienie przełącznika ostrzeżeń	186
\$^X	— nazwa pliku interpretera	186
\$	— zmienna domyślna	186
\$`	— tekst przed wzorcem	186
\$	— czyszczenie bufora wyjściowego	187
\$~	— nazwa formatu raportowania	187
\$+	— ostatnie dopasowanie nawiasów	187
\$<	— rzeczywisty UID	187
\$=	— obowiązująca długość strony	187
\$>	— obowiązujący UID	188
\$0	— nazwa programu	188
\$ARGV	— nazwa bieżącego pliku	188
\$n	— dopasowanie numer <i>n</i>	188
%ENV	— ustawienia środowiska	189
%INC	— pliki włączane	189
%SIG	— obsługa sygnałów	189
@_	— parametry procedur	189
@ARGV	— parametry wiersza poleceń	189
@INC	— położenie uruchamianych skryptów	190

Rozdział 10. Wbudowane funkcje: przetwarzanie danych 191

W skrócie	191
Funkcje wbudowane	191
Zagadnienia	192
abs — wartość bezwzględna	192
atan2 — arcus tangens	192
chomp — usunięcie końców wierszy	192
chop — usunięcie ostatniego znaku	193
chr — znak o podanym kodzie	193
cos — cosinus	193
each — pary kluczy i wartości asocjacji	194
eval — uruchamianie kodu Perla	194
exists — sprawdzanie istnienia klucza w asocjacji	194
exp — potęga liczby <i>e</i>	194
hex — konwersja na liczbę szesnastkową	195
index — położenie części tekstu	195
int — odrzucenie części ułamkowej	195
join — złączenie elementów listy w ciąg znaków	196
keys — klucze asocjacji	196
lc — zmiana na małe litery	196

lcfirsr — zmiana pierwszego znaku na małą literę	196
length — długość tekstu	197
log — logarytm naturalny	197
map — wykonanie kodu dla każdego elementu	197
oct — konwersja wartości ósemkowych	197
ord — kod ASCII	198
pack — kompresja danych	198
pop — odebranie z tablicy elementu	199
POSIX (grupa funkcji)	199
push — wstawienie wartości do tablicy	200
rand — liczba losowa	200
reverse — odwrócenie kolejności elementów listy	201
rindex — funkcja odwrotna do index	201
scalar — wymuszenie kontekstu skalarne go	201
shift — przesunięcie wartości w tablicy	201
sin — sinus	202
sort — sortowanie listy	202
splice — warstwy tablic	203
split — rozbić tekst na tablicę znaków	203
sprintf — formatowanie tekstu	203
sqr — pierwiastek kwadratowy	205
srand — wartość inicjalizująca generatora liczb losowych	206
substr — zwraca część tekstu	206
time — sekundy od 1 stycznia 1970 roku	206
uc — wielkie litery	207
ucfirst — zmiana pierwszej litery na wielką	207
unpack — rozpakowywanie danych	207
unshift — przesuwanie wartości w tablicy z ich dodawaniem	207
values — wartości asocjacji	208
vec — wektor składający się z liczb całkowitych bez znaku	208
Rozdział 11. Funkcje wbudowane: I/O i komunikacja między procesami	209
W skrócie	209
Formaty Perla	209
Zagadnienia	210
print — drukowanie danych listowych	210
printf — drukowanie formatowanych danych z list	211
Użycie <> do odczytu danych	212
getc — pobranie znaku	212
write — zapis sformatowanego rekordu	212
Formaty: wyrównanie tekstu do lewej	213
Formaty: wyrównanie tekstu do prawej	213
Formaty: centrowanie tekstu	213
Formaty: prezentacja liczb	214
Formaty: formatowanie danych wielowierszowych	214
Formatowanie: rozdzielanie tekstu do formatowania danych wielowierszowych	214
Formatowanie: wielowierszowe dane bez formatowania	215
Formaty: dane w nagłówku raportu	215
Formaty: zmienne formatów	216
warn — wyświetlenie ostrzeżenia	216
IPC: exec — wywołanie polecenia systemowego	217
IPC: system — rozgałęzienie procesów i uruchomienie polecenia systemowego	218
IPC: odczytywanie danych z innego programu	218
IPC: wysyłanie danych do innego programu	219

IPC: zapisywanie danych do procesu potomnego	219
IPC: zapisywanie danych do procesu rodzica	220
IPC: wysyłanie sygnału do innego procesu	221
IPC: użycie gniazd	222
IPC: OLE w Win32	223
Rozdział 12. Funkcje wbudowane: obsługa plików.....	225
W skrócie	225
Wszystko o obsłudze plików	225
Zagadnienia	226
open — otwarcie pliku.....	226
close — zamknięcie pliku.....	228
print — drukowanie do pliku.....	228
write — pisanie do pliku.....	229
binmode — ustawienie trybu binarnego.....	229
Ustawianie buforowania kanału wyjściowego	230
Odczyt plików przekazanych w wierszu poleceń	230
Odczyt danych z uchwytu pliku	231
read — odczyt danych	231
readline — odczyt wiersza danych	232
getc — pobranie znaku	232
seek — ustawianie położenia w pliku	232
tell — określanie położenia w pliku	233
stat — status pliku	234
Funkcje obsługi plików w POSIX	235
select — określanie domyślnego uchwytu pliku wyjścia	235
eof — sprawdzanie końca pliku	236
Zapisywanie pliku bazy danych DBM	237
Czytanie pliku bazy danych DBM.....	237
flock — blokowanie pliku	238
chmod — zmiana uprawnień do pliku.....	238
glob — wyszukanie pasujących plików	238
rename — zmiana nazwy pliku	239
unlink — usuwanie plików	239
opendir — otwarcie katalogu.....	239
closedir — zamknięcie katalogu.....	240
readdir — odczytanie pozycji katalogu	240
seekdir — ustawienie się w katalogu.....	240
telldir — określenie położenia w katalogu	240
rewinddir — ustawienie się na początku katalogu	240
chdir — zmiana katalogu roboczego	240
mkdir — utworzenie katalogu	241
rmdir — usunięcie katalogu.....	241
Część III Programowanie w Perlu.....	243
Rozdział 13. Moduły wbudowane	245
W skrócie	245
Użycie modułów Perla.....	245
Zagadnienia	251
Term::Cap — obsługa terminala	251
Math: duże liczby i liczby urojone	251
POSIX: Przenośny interfejs dostępu do systemu operacyjnego	252
Benchmark: testy wydajności	253
Time: czas i jego przeliczanie.....	253

Carp: zgłaszanie błędów z punktu widzenia wywołującego	253
locale: ustawienia lokalne	254
File: operacje plikowe	254
Net: dostęp do Internetu	254
Safe: bezpieczne wykonywanie kodu	255
Tk: zestaw narzędzi Tk	255
Tk: przycisk i pole tekstowe	255
Tk: przyciski radio i pole opcji	256
Tk: lista wyboru	256
Tk: suwak	257
Tk: płótno	257
Tk: menu	258
Tk: Okienka dialogowe	259
Rozdział 14. Struktury danych	261
W skrócie	261
Żądanie deklarowania zmiennych? Świetny pomysł!	264
Zagadnienia	264
Złożone rekordy: zapisywanie wskaźników i innych elementów	264
Deklarowanie tablic tablic	265
Tworzenie tablic tablic na bieżąco	266
Sięganie do tablic tablic	267
Deklarowanie asocjacji asocjacji	268
Tworzenie asocjacji asocjacji na bieżąco	268
Sięganie do asocjacji asocjacji	269
Deklarowanie tablic asocjacji	270
Tworzenie tablic asocjacji na bieżąco	270
Sięganie do tablic asocjacji	271
Deklarowanie asocjacji tablic	272
Tworzenie asocjacji tablic na bieżąco	272
Sięganie do asocjacji tablic	272
Listy powiązanych wartości i bufory pierścieniowe	273
Rozdział 15. Tworzenie pakietów i modułów	275
W skrócie	275
Pakiety	275
Moduły	277
Zagadnienia	277
Tworzenie pakietu	277
BEGIN — konstruktor pakietu	279
END — destruktor pakietu	279
Określanie bieżącego pakietu	280
Pakiet w wielu plikach	280
Tworzenie modułów	281
Domyślne eksportowanie symboli z modułów	281
Udostępnianie symboli z modułu do eksportu	282
Blokowanie importowania symboli	283
Blokowanie eksportowania symboli	283
Eksportowanie bez metody import	284
Submoduły zagnieżdżone	284
Sprawdzanie numeru wersji modułu	285
Automatyczne ładowanie procedur	286
AutoLoader i SelfLoader	287

Rozdział 16. Klasy i obiekty	289
W skrócie	289
Klasy	290
Obiekty	290
Metody	291
Dziedziczenie	291
Zagadnienia	292
Tworzenie klasy	292
Tworzenie konstruktora	292
Tworzenie obiektu	292
Tworzenie metody klasy	293
Tworzenie instancji metody	293
Wywoływanie metody	294
Tworzenie zmiennej instancji	294
Tworzenie metod i zmiennych prywatnych	295
Tworzenie zmiennej klasy	296
Tworzenie destruktora	296
Dziedziczenie klas	297
Dziedziczenie konstruktorów	298
Dziedziczenie zmiennych instancji	299
Dziedziczenie wielokrotne	299
Przykrywanie metod klasy bazowej	300
Wiązanie skalarów	301
Wiązanie tablic	302
Wiązanie asocjacji	303
Użycie klasy UNIVERSAL	303
Rozdział 17. Usuwanie błędów z kodu Perla. Wytyczne dla programistów	305
W skrócie	305
Przykładowa sesja usuwania błędów	306
Zagadnienia	307
Przechwytywanie błędów wykonania	307
Uruchamianie debuggera	308
Dostępne polecenia debuggera	308
Prezentacja kodu	309
Praca „krok po kroku”	310
Przeskakiwanie wywołań procedur	310
Ustawianie punktów kontrolnych	311
Usuwanie punktów kontrolnych	311
Wykonywanie kodu do punktu kontrolnego	312
Wyświetlanie wyrażenia	312
Wyliczanie wartości wyrażenia	312
Zmiana wartości zmiennych	312
Ustawianie podglądu	313
Realizacja zadań debuggera	313
Kończenie pracy z debuggerem	314
Wytyczne dla programistów Perla	314
Część IV Programowanie CGI	317
Rozdział 18. Programowanie CGI	319
W skrócie	319
Użycie CGI.pm	320
Tworzenie i używanie elementów graficznych HTML	321
Zagadnienia	326

Użycie PerlScriptu	326
Początek dokumentu HTML w CGI.....	327
Tworzenie nagłówków HTML	327
Wyśrodkowanie elementów HTML	328
Tworzenie list HTML	328
Tworzenie hiperłączy.....	328
Tworzenie poziomej linii	329
Tworzenie formularza HTML	329
Używanie pól tekstowych.....	329
Odczytywanie danych spod elementów graficznych HTML	330
Używanie obszarów tekstowych.....	330
Używanie pól opcji	331
Używanie list wyboru	331
Używanie przycisków radio	332
Używanie menu	332
Użycie pól ukrytych.....	333
Tworzenie przycisków SUBMIT i RESET	333
Zakończenie formularza HTML	333
Zakończenie dokumentu HTML.....	334
Programowanie CGI przy pomocy funkcji.....	334
Rozdział 19. Programowanie CGI przy użyciu cgi-lib.pl.....	337
W skrócie	337
Użycie cgi-lib.pl	339
Zagadnienia	342
Jakie procedury zawiera cgi-lib.pl?	342
Początek dokumentu HTML.....	343
Tworzenie nagłówków HTML	344
Wyśrodkowanie elementów HTML	344
Tworzenie list HTML	344
Tworzenie hiperłączy.....	344
Tworzenie poziomej linii	345
Tworzenie formularza HTML	345
Używanie pól tekstowych.....	345
Odczytywanie danych spod elementów graficznych HTML	346
Używanie obszarów tekstowych.....	346
Używanie pól opcji	346
Używanie list wyboru	347
Używanie przycisków radio	348
Używanie menu	348
Użycie pól ukrytych.....	349
Tworzenie przycisków SUBMIT i RESET	349
Zakończenie formularza HTML	349
Zakończenie dokumentu HTML.....	349
Wyświetlanie wszystkich zmiennych	350
Rozdział 20. CGI: liczniki odwiedzin, księga gości, wysyłanie listów, kwestie bezpieczeństwa	351
W skrócie	351
Bezpieczeństwo CGI	352
Zagadnienia	352
Bezpieczeństwo na poważnie	352
Użycie błędnych danych.....	354
Korygowanie danych.....	355

Zwiększanie uprawnień skryptu CGI w systemie UNIX	356
Tworzenie licznika odwiedzin	356
Tworzenie książki gości	357
Użycie skryptów CGI do wysyłania poczty	362
Rozdział 21. Pogawędki (chat), cookies i gry.....	367
W skrócie	367
Aplikacje do pogawędek.....	367
Cookies	368
Gra	368
Zagadnienia	368
Aplikacja do pogawędek.....	368
Zapisywanie i odczytywanie cookies	374
Tworzymy grę.....	377
Dodatki	385
Dodatek A Przewodnik.....	387
Wartość prawdy — true	387
Wyróżniki przedrostkowe zmiennych	387
Zmienne skalarne	388
Listy	388
Kontekst skalarny a kontekst listowy.....	388
Tablice.....	389
Asocjacje	389
Typy ogólne	390
Operatory	390
Przypisywanie wartości.....	390
Operatory porównania.....	390
Operatory równości.....	391
Instrukcja if	393
Instrukcja unless.....	393
Instrukcja for	393
Instrukcja foreach.....	393
Instrukcja while	394
Użycie pętli until	394
Modyfikowanie instrukcji za pomocą fraz if, unless, until i while	394
Instrukcje pętli.....	394
Instrukcja goto.....	395
Procedury	395
Odczyt parametrów przekazanych procedurze	396
Zwracanie wartości z procedur	396
Wskaźniki.....	396
Dereferencja wskaźników	397
Zmienne wbudowane	397
Pakiety.....	398
Skorowidz	399

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 = [
    ["jabłko", "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:

<code>\$array[1][2]</code>	<code># tablica tablic</code>
<code>\$hash{bigkey}{littlekey}</code>	<code># asocjacja asocjacji</code>
<code>\$array[3]{key}</code>	<code># tablica asocjacji</code>
<code>\$hash{key}[4]</code>	<code># asocjacja tablic</code>

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' => 'jabłko', '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
jabłko
1
jabłko
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 = (
    ["jabłko", "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 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 => "jabłko",
        'też dobre' => "pomarańcze",
    },
    warzywa => {
        ulubione => "kukurydza",
        'też dobre' => "groszek",
        'nie lubiane' => "rzepa",
    },
);
print $hash{owoce}{też dobre};

pomarańcze

```

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

```

%hash = (
    owoce => {
        ulubione => "jabłko",
        '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' => 'jabłko' 'też dobre' => 'pomarańcze'}

```

Do uporządkowania pierwszego poziomu asocjacji należy użyć następującego wyrażenia:

```

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

```

Deklarowanie tablic asocjacji

Tablicę asocjacji stosujemy do indeksowania rekordów oznaczonych kluczami (przykład zobaczmy 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",  
    'nie lubiane' => "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", 'nie lubiane' => "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", 'nie lubiane' => "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,  
           nie lubiane: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 => "jabłko",
             'też dobre' => "pomarańcze"};
$array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nie lubiane' => "rzepa"};
$array[2] = {ulubione => "kurczak",
             'też dobre' => "wołowina"};
print $array[0]{ulubione};

jabłko

```

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

```

$array[0] = {ulubione => "jabłko",
             'też dobre' => "pomarańcze"};
$array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nie lubiane' => "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' => 'jabłko' 'też dobre' => 'pomarańcze' };
array[1] = {'ulubione' => 'kukurydza' 'też dobre' => 'groszek'
            'nie lubiane' => 'rzepa' }
array[2] = {'ulubione' => 'kurczak' 'też dobre' => 'wołowina' }

```

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

```

$array[0] = {ulubione => "jabłko",
             'też dobre' => "pomarańcze"};
$array[1] = {ulubione => "kukurydza", 'też dobre'
             => "groszek", 'nie lubiane' => "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 => ["jabłko", "pomarańcze"],
    warzywa => ["kukurydza", "groszek", "rzepa"],
    'mięso' => ["kurczak", "szynka"],
);
print $hash{owoce}[0];

jabłko
```

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

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

owoce: [jabłko, 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 w 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.