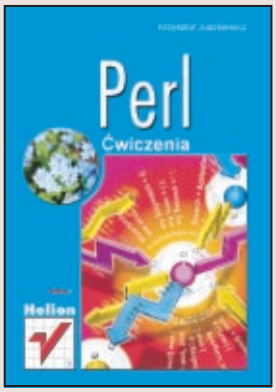
Perl. Ćwiczenia praktyczne



Autor: Krzysztof Juszkiewicz
ISBN: 83-7197-946-0
Format: B5, stron: 132

Jeśli chcesz rozpocząć przygodę z programowaniem – zacznij naukę od Perla. Po kilku godzinach pracy przekonasz się, że dokonałeś najlepszego wyboru. Perl posiada wielką zaletę, której brakuje konkurencyjnym narzędziom: nie znając wszystkich jego niuansów możesz pisać działające aplikacje.

Larry Wall stworzył Perla dla osób, które są niecierpliwe (programy w Perlu pisze się bardzo szybko), leniwe (kod programów jest zwięzły, a ich tworzenie ułatwia ogromna liczba gotowych modułów) i chcą być dumne z tego, co robią (sam przekonasz się, jak wiele satysfakcji dostarczy Ci programowanie w Perlu).

Książka przeznaczona jest dla początkujących programistów. Dowiesz się z niej jak zainstalować Perla, a następnie, krok po kroku, ćwiczenie po ćwiczeniu, zagłębisz się w jego świat.

Poznasz:

- Fundamenty języka: zmienne, instrukcje i referencje
- Wyrażenia regularne, umożliwiające szybkie przeszukiwanie tekstów i podmianę fragmentów
- Sposoby korzystania z plików
- Zasady pisania skryptów CGI
- Wysyłanie e-maili za pomocą Perla
- Łączenie Perla z bazami danych

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

Spis treści

Rozdział 7.

Wyrażenia regularne

Język Perl jest dobrze przystosowany do przetwarzania tekstów. Jego główną zaletą wiążącą się z tym zagadnieniem są wyrażenia regularne. Ich istotą jest tworzenie wzorców opartych o różne symbole. Taki wzorzec jest dopasowywany do testowanego tekstu i w ten sposób można sprawdzać, czy dany tekst zawiera określone litery, znaki, grupy liter, słowa. Opierając się na wykonanym dopasowaniu (lub jego braku) można dodatkowo podmienić pasującą frazę na inną.

Sprawdzanie

Ćwiczenie 7.1.

Sprawdź, czy w zmiennej region znajduje się słowo karp.

Rozwiązanie

```
#!/usr/bin/perl
$region = "podkarpacki";
if ($region =~ /karp/) {
  print "Jest karp\n";
}
```

Wartość zmiennej `region` ustawiamy na `podkarpacki`. W tym słowie znajduje się fraza `karp`, więc efekt sprawdzenia będzie pozytywny i na ekranie powinniśmy zobaczyć napis wygenerowany przez funkcję `print`.

Do obsługi wyrażeń regularnych używamy operatora składającego się ze znaku równości i tyldy (=~). W niektórych edytorach mogą wystąpić problemy z wpisaniem znaku tyldy (~), gdyż po naciśnięciu odpowiedniego klawisza nie pojawia się on na ekranie. Należy wówczas oprócz właściwego klawisza użyć spacji.

Po operatorze wyrażenia regularnego piszemy samo wyrażenie ujęte w ukośniki (/). Wpisanie ciągu liter (tak jak w tym przypadku) spowoduje sprawdzenie, czy w danej zmiennej zawierającej ciąg znaków (`podkarpacki`) występuje zadany podciąg (`karp`).

Ćwiczenie 7.2.

Zbadaj, czy w zmiennej region nie ma słowa karp.

Rozwiązanie

```
#!/usr/bin/perl
if($region !~ /karp/) {
  print "Brak karp\n";
}

if(! $region =~ /karp/) {
  print "Brak karp\n";
}

unless($region =~ /karp/) {
  print "Brak karp\n";
}
```

W tym przypadku przedstawiono trzy alternatywne rozwiązania. Są one równoważne. W pierwszym wierszu widzimy, że zastosowano operator `!~`. Sprawdzamy zatem, czy gdziekolwiek w zmiennej `region` nie ma układu liter `karp`. Ten sam efekt uzyskamy stosując wykrzyknik przed warunkiem (warunek drugi) oraz dzięki użyciu warunku negatywnego `unless` (jeśli nie).

Ćwiczenie 7.3.

Prześledź, czy słowo karp znajduje się w zmiennej domyślnej.

Rozwiązanie

```
#!/usr/bin/perl
$_ = "podkarpacki";
if(/karp/) {
  print "Jest karp\n";
}
```

Zmienna domyślna to taka, której nazwa składa się jedynie z podkreślenia (_). Jeśli w takiej zmiennej umieścimy nasz tekst do sprawdzenia (`podkarpacki`), nie trzeba stosować operatora wyrażenia regularnego (=~), wystarczy jedynie samo wyrażenie w ukośnikach.

Ćwiczenie 7.4.

Przeanalizuj, czy odpowiedź zawarta w zmiennej odp jest twierdząca.

Rozwiązanie

```
#!/usr/bin/perl
if($odp =~ /tak/) {
  print "Jest OK\n";
}
```

Sprawdzamy tutaj za pomocą wyrażenia regularnego, czy odpowiedź zawiera słowo `tak`. Nie możemy mieć jednak pewności, czy odpowiedzią (wartość zmiennej `odp`) nie była na przykład `taktyka` lub `taknie`. Powinniśmy zatem zastosować jeden z poniższych wierszy.

```
#!/usr/bin/perl
if($odp eq "tak") {
  print "Jest OK\n";
}
if ($odp =~ /^tak$/) {
  print "Jest OK\n";
}
```

W pierwszym wierszu widzimy porównanie (`eq`) wartości zmiennej `odp` i ciągu znaków `tak`, natomiast w drugim warunku mamy do czynienia z adekwatnym wyrażeniem regularnym. Zastosowanie daszka (`^`) i dolara (`$`) oznacza w tym przypadku odpowiednio początek i koniec ciągu znaków. Innymi słowy, chcemy by słowo tak było dokładnie tym, co znajduje się w zmiennej `odp`. Właśnie po to stosujemy znaki daszka i dolara.

Istnieje jednak nadal możliwość, że odpowiedź była twierdząca, a my tego nie stwierdzimy. Stanie się tak, gdy zmienna `odp` będzie miała przykładowo wartość `TAK` lub `Tak`. Zaradzimy takiemu zjawisku używając innej konstrukcji.

```
#!/usr/bin/perl
if ($odp =~ /^tak$/i) {
  print "Jest OK\n";
}
```

Różni się ona od poprzedniej literą `i`, występującą po drugim ukośniku wyrażenia regularnego. Dzięki temu przy porównywaniu duże i małe litery traktowane są tak samo, czyli słowa `TAK` i `tak` są równoważne.

Ćwiczenie 7.5.

Sprawdź, czy w zmiennej adres występuje cyfra 2.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 2 m. 39";

if($adres =~ /2/) {
  print "Jest dwa\n";
}
```

Wartości liczbowe w języku Perl traktowane są tak samo jak znaki, dlatego w wyrażeniach regularnych możemy używać również cyfr, w taki sam sposób jak liter.

Ćwiczenie 7.6.

Ustal, czy w adresie występuje ukośnik.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 2/39";

if($adres =~ /\//) {
  print "Mieszkanie w bloku\n";
}
```

Wyrażenia regularne umieszczamy w ukośnikach (`//`). Jeśli chcemy umieścić ukośnik w tak zapisanym wyrażeniu regularnym, musimy go poprzedzić odwrotnym ukośnikiem (`\`). W ten sposób zapis (`/\//`) staje się dość nieczytelny. Możemy jednak zmienić sposób zapisu wyrażenia, zmieniając znaki ograniczające to wyrażenie (`//`) na dowolne inne (z wyjątkiem liter i cyfr), np. na kreskę pionową (`|`).

```
#!/usr/bin/perl
$adres = "Bajkowa 2/39";

if($adres =~ m|/|) {
  print "Mieszkanie w bloku\n";
}
```

Warunkiem zmiany ograniczającego ukośnika na inny znak jest umieszczenie przed wyrażeniem litery `m`. Czasem można też spotkać zapis wyrażenia regularnego z ukośnikami i literą `m` (`m//`). Nie jest on błędny, lecz — jak widać w poprzednich ćwiczeniach — nadmiarowy.

Ćwiczenie 7.7.

Sprawdź, czy numer domu lub mieszkania jest równy 2.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 2 m. 32";

if($adres =~ / 2 /) {
  print "Jest dwa\n";
}
```

Za zadanie mamy sprawdzić, czy w zmiennej zawierającej adres występuje liczba `2`. Spodziewamy się, że mogą wystąpić tu dwie liczby, z których przynajmniej jedna jest równa dwa. Nie można jako wyrażenia regularnego w tym przypadku użyć `/2/`, gdyż moglibyśmy wówczas jako dwa potraktować liczbę, której jedną z cyfr jest dwa (np. `32`). Zastosujemy zatem sprawdzanie, czy w zmiennej `odp` znajduje się fraza składająca się z dwóch spacji, pomiędzy którymi występuje cyfra `2`.

Przedstawione rozwiązanie nie jest jednak tym, które spełnia warunki zadania. Poprawny zapis wyrażenia regularnego znajduje się poniżej.

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

if($adres =~ /\b2\b/) {
  print "Jest dwa\n";
}
```

Widzimy tu, że niekoniecznie przed oraz za liczbą dwa musi wystąpić spacja. W tej sytuacji zamiast spacji należy zastosować ograniczenie słowa (`\b`). Jest to analogiczna konstrukcja, jak w przypadku początku i końca frazy (`^$`).

Ćwiczenie 7.8.

Zbadaj, czy któryś z numerów w adresie jest parzysty.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

if($adres =~ /[02468]\b/) {
  print "Jest parzysta\n";
}
```

Liczba parzysta to taka, która kończy się cyfrą parzystą, czyli jedną ze zbioru: 0, 2, 4, 6, 8. Rozwiązanie jest bardzo podobne do poprzedniego, z tą różnicą, że mamy odpowiedzieć na pytanie o ostatnią cyfrę. Dlatego występuje tu tylko jedno ograniczenie końca wyrazu `\b`.

Zamiast sprawdzania, czy wystąpiła tylko jedna cyfra, sprawdzamy, czy nie wystąpił jeden z pięciu wymienionych znaków (w tym przypadku cyfr). Robimy to przy użyciu oznaczenia klasy znaków w języku Perl, używając nawiasów kwadratowych `[]`.

Ćwiczenie 7.9.

Prześledź, czy w adresie nie występuje liczba nieparzysta.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

if($adres !~ /[13579]\b/) {
  print "Brak nieparzystej\n";
}
```

Analogicznie jak poprzednio, musimy za pomocą `\b` ustalić, jaki jest ostatni znak wyrazu i czy nie jest on nieparzysty.

Ćwiczenie 7.10.

Jeśli w adresie jest liczba parzysta, znajdź ją.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

if($adres =~ /\d*[02468]\b/) {
  print "Jest parzysta: $&\n";
}
```

W poprzednim ćwiczeniu używaliśmy wyrażenia `[02468]\b`. Teraz dodaliśmy `\d*`, a także ujęliśmy wszystko w nawiasy.

Kombinacja `\d` oznacza wystąpienie dowolnej cyfry i jest równoznaczna z zapisem `[0123456789]`, który w skrócie wygląda następująco — `[0-9]`.

Gwiazdka (*) w wyrażeniach regularnych oznacza, że poprzedzające ją wyrażenie nie musi wcale wystąpić lub może się pojawić wielokrotnie (czyli 0 lub więcej razy). W naszym przypadku przed ostatnią cyfrą parzystą może pojawić się w liczbie wiele innych cyfr, co nie będzie miało znaczenia dla parzystości szukanej liczby.

Jeśli w adresie jest liczba parzysta, jej wartość znajdzie się w zmiennej `$&`, którą następnie wyświetlamy. Ciąg znaków poprzedzający dopasowanie znajdzie się w `$``, zaś występujący po — w zmiennej `$'`. Interpreter zawsze znajduje w takiej sytuacji tylko pierwsze miejsce, w którym spełniony jest zadany warunek.

Ćwiczenie 7.11.

Znajdź obie lub jedną liczbę parzystą w adresie.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

if($adres =~ /(\d*[02468]\b)\D*(\d*[02468]\b)?/) {
  print "Jest parzysta: $1 $2\n";
}
```

Wyrażenie regularne ma tu postać podobną jak poprzednio, lecz powtórzone zostały dwukrotnie sekwencje `\d*[02468]\b`. Obie ujęto w nawiasy, a po drugiej dodano znak zapytania. Dzięki niemu ta druga sekwencja oznaczająca liczbę parzystą może, lecz nie musi, wystąpić, co pozwala nam znaleźć jedną lub obie liczby parzyste, w zależności od tego, ile ich wystąpiło.

Pomiędzy liczbami powinny znaleźć się jakieś znaki nie będące cyframi (symbol `\D`), o ile w adresie są dwie liczby.

Zastosowanie nawiasów w wyrażeniu regularnym spowoduje, że w przypadku, gdy sprawdzanie ciągu się powiedzie (u nas — gdy w adresie wystąpi co najmniej jedna liczba parzysta), po sprawdzeniu ustawione zostaną zmienne, których nazwy są liczbami. Kolejne zmienne odpowiadają kolejności użytych nawiasów. Efektem działania programu będzie napis:

```
Jest parzysta: 32 2
```

Wartości zmiennych o nazwach liczbowych są kasowane przed użyciem kolejnego wyrażenia regularnego, dlatego należy ich wartości zachowywać w innych zmiennych, na przykład za pomocą następującego zapisu:

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

$adres =~ /(\d*[02468]\b)\D*(\d*[02468]\b)?/;
@parzyste = ($1,$2);
```

Można także zastosować konstrukcję, która wynik umieści wprost w tablicy:

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

@parzyste = $adres =~ /(\d*[02468]\b)\D*(\d*[02468]\b)?/;
```

Ćwiczenie 7.12.

Wyświetl kolejno wszystkie liczby parzyste w adresie.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

while ($adres =~ /(\d*[02468]\b)/g) {
  print "$1\n";
}
```

Wartość zmiennej `adres` jest dopasowywana do znanego nam już wzorca `\d*[02468]`. Tym razem jednak po wyrażeniu regularnym pojawiła się litera `g`, oznaczająca konieczność przeszukania całego ciągu, nie zaś — jak poprzednio — tylko pierwszego miejsca, które pasuje do wzorca. Na ekranie zobaczymy następujące linie wygenerowane przez program:

```
32
2
```

Ćwiczenie 7.13.

Pokaż wszystkie liczby znajdujące się na początku kolejnych wierszy.

Rozwiązanie

```
#!/usr/bin/perl
$wiersze = "12\n34 18\n373\n";

while ($wiersze =~ /^(\d+)/gm) {
  print "$1\n";
}
```

W zmiennej `wiersze` umieszczono cztery liczby w trzech wierszach (trzy znaki nowej linii). Wyrażenie `^(\d+)` będzie pasować jedynie do jednej liczby znajdującej się na początku zmiennej, nawet w przypadku zastosowania modyfikatora `g`, użytego w poprzednim

ćwiczeniu. Dzieje się tak, ponieważ znaki ^ i $ oznaczają końce wartości całej zmiennej. Dodanie modyfikatora m spowoduje, że wspomniane znaki odnosić się będą do podziału wyznaczonego przez końce linii. Wydruk na ekranie będzie się przedstawiał następująco:

```
12
34
373
```

Ćwiczenie 7.14.

Znajdź czwarty znak w zmiennej adres.

Rozwiązanie

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

($znak) = ($adres =~ /.{3}(.)/);
print $znak;
```

W wyrażeniu regularnym widzimy kropkę, która zastępuje dowolny znak. Po niej w nawiasach klamrowych jest liczba 3, oznaczająca, że poprzedzający ją znak musi wystąpić dokładnie trzy razy. Następnie znowu jest kropka, w zwykłych nawiasach. Dzięki temu po dopasowaniu do wyrażenia regularnego ustawiona zostanie zmienna $1 o wartości równej czwartemu znakowi. Ponadto ten znak zostanie zwrócony i podstawiony do zmiennej znak, a na ekranie zobaczymy literę k.

Zastosowanie wyrażenia regularnego do tego typu problemu nie jest najlepszym rozwiązaniem. Lepiej użyć funkcji substr.

```
#!/usr/bin/perl
$adres = "Bajkowa 32/2";

$znak = substr($adres,3,1);
```

Ćwiczenie 7.15.

Znajdź wszystkie cyfry znajdujące się w zmiennej liczba przed cyfrą 0.

Rozwiązanie

```
#!/usr/bin/perl
$liczba = "34430321042";
print $liczba =~ /(\d*)0/;
```

W zmiennej liczba cyfra zero występuje dwukrotnie. Stosując wzorzec (\d*) otrzymujemy największe możliwe dopasowanie, czyli zawierające pierwsze zero. Efektem działania funkcji print będzie liczba 34430321.

Gdy chcemy otrzymać najmniejsze dopasowanie, należy po gwiazdce dodać znak zapytania. W naszym przypadku, jeśli chcemy otrzymać liczbę 3443, wzorzec powinien mieć postać /(\d*?)0/.

Ten sam rezultat można uzyskać dzięki wyrażeniu `/([1-9]*)/`. Oznacza ono zero lub wielokrotne (*) wystąpienie cyfr od `1` do `9`.

Ćwiczenie 7.16.

Sprawdź, czy w zmiennej region jest ciąg liter umieszczony w zmiennej fragment.

Rozwiązanie

```
#!/usr/bin/perl
$region = "podkarpacki";
$fragment = "karp";

if($region =~ /$fragment/) {
  print "Jest ciąg $fragment\n";
}
```

W pierwszych dwóch wierszach ustawiamy wartości zmiennych `region` i `fragment` odpowiednio na `podkarpacki` oraz `karp`. Na początku tego rozdziału zastosowaliśmy warunek `$region =~ /karp/`. Teraz zamiast słowa `karp` stosujemy wartość zmiennej `fragment`, która go zawiera.

Takie zastosowanie wyrażenia regularnego jest dość wygodne, lecz może wydłużyć czas wykonywania skryptu, gdyż przed każdym dopasowaniem wyrażenia regularnego (na przykład w pętli) interpreter musi sprawdzić, czy wartość zmiennej `fragment` nie uległa zmianie. Wiedząc, że wartość zmiennej nie ulega zmianie, programista może wyłączyć sprawdzanie dodając literę o po wyrażeniu regularnym, tak by ostatni wiersz miał następującą postać:

```
#!/usr/bin/perl
if($region =~ /$fragment/o) {
  print "Jest ciąg $fragment\n";
}
```

Ćwiczenie 7.17.

Do zmiennej fragment wstaw nawias i sprawdź, czy występuje on w zmiennej region.

Rozwiązanie

```
#!/usr/bin/perl

$region = "(pod)karpacki";
$fragment = '\(';

if($region =~ /$fragment/o) {
  print "Jest ciąg $fragment\n";
}
```

Nawias w wyrażeniu regularnym ma znaczenie grupujące, dlatego należy go poprzedzić odwrotnym ukośnikiem. Jego pominięcie spowoduje błąd, wyglądający jak poniżej:

```
/(/: unmatched () in regexp
```

Użycie odwrotnego ukośnika jest nieeleganckie, gdyż funkcja `print` da na ekranie mylący efekt.

```
Jest ciąg \(
```

Innym sposobem jest wyłączenie znaczenia znaków specjalnych w wyrażeniu regularnym. Po modyfikacji rozwiązanie będzie miało następującą postać:

```
#!/usr/bin/perl
$region = "(pod)karpacki";
$fragment = '(';
if($region =~ /\Q$fragment\E/o) {
  print "Jest ciąg $fragment\n";
}
```

Wspomniane wyłączenie będzie miało miejsce od znaków `\Q` do znaków `\E`. Można te sekwencje stosować wielokrotnie.

Zamiana

Ćwiczenie 7.18.

Zamień w zmiennej wladca słowo Kazimierz na Stefan.

Rozwiązanie

```
#!/usr/bin/perl
$wladca = "Kazimierz Batory";
$wladca =~ s/Kazimierz/Stefan/;
```

Za pomocą wyrażeń regularnych możemy nie tylko sprawdzić, czy dane wyrażenie występuje, lecz również dopasowane wyrażenie zastąpić innym. Służy do tego operator `s`, analogiczny do operatora sprawdzania — `m`.

W przypadku zamiany z wykorzystaniem wyrażenia regularnego należy podać dwa wzorce, jeden do dopasowania (u nas `Kazimierz`), drugi do zamiany (`Stefan`). Po zamianie wartością zmiennej `wladca` będzie `Stefan Batory`.

Ćwiczenie 7.19.

Usuń ze zmiennej znaczki wszystkie białe znaki na początku i końcu jej wartości.

Rozwiązanie

```
#!/usr/bin/perl
$znaczki = "  tekst testowy  ";

$znaczki =~ s/^\s+//;
$znaczki =~ s/\s+$//;
print $znaczki;
```

W pierwszym wierszu usuwamy białe znaki (`\s`) występujące na początku (`^`) zmiennej `znaczki`. Plus (`+`) oznacza konieczność co najmniej jednokrotnego wystąpienia któregoś z białych znaków. Może to być spacja, tabulacja lub znak nowej linii. Analogicznie w drugim wierszu pozbywamy się znaków przed końcem (`$`) wartości zmiennej. Białe znaki w środku tekstu pozostaną niezmienione, a na ekranie zobaczymy:

```
tekst testowy
```

Istnieją również dwa różne sposoby realizacji tego zadania, przy zastosowaniu zapisu jednolinijkowego. Są one jednakże wolniejsze w działaniu niż poprzednie rozwiązanie.

```
#!/usr/bin/perl
$znaczki = "  tekst testowy  ";

$znaczki =~ s/^\s*(.*?)\s*$/$1/;
```

W związku z tym, że należy przyjąć, iż białe znaki mogą wystąpić jedynie z jednej strony, należy zamiast plusa (`+`) użyć gwiazdki (`*`). W ten sposób dopuszczamy brak wystąpienia białego znaku.

W pierwszej wersji korzystamy z zamiany pewnej ograniczonej liczby (`*?`) dowolnych znaków (`.`) na te same znaki, ale bez otaczających białych znaków. Zamiana jest dokonywana poprzez zastosowanie nawiasów i wykorzystanie wartości zmiennej tymczasowej (`$1`).

```
#!/usr/bin/perl
$znaczki = "  tekst testowy  ";

$znaczki =~ s/^\s*|\s*$//g;
```

Drugi sposób opiera się na zamianie białych znaków na nic. Dodatkowo sprawdzane jest, czy występują one na początku lub (`|`) na końcu zmiennej.

Ćwiczenie 7.20.

Zamień znaki końca linii sytemu uniksowego na znaki końca linii w systemie Windows.

Rozwiązanie

```
$znaczki =~ s/\n/\r\n/gm;
```

Dla systemu uniksowego znak końca linii w języku Perl zapisujemy za pomocą wyrażenia `\n`, natomiast dla Windows są to dwa znaki — `\r\n`. Należy pamiętać, że w tym przypadku jeden znak zastępujemy dwoma, z których jeden to ten sam znak. Modyfikatory (litery na końcu) `gm` są konieczne w przypadku występowania w zmiennej `znaczki` wielu znaków nowej linii.

Ćwiczenie 7.21.

Zamień numery miesięcy na ich nazwy.

Rozwiązanie

```
#!/usr/bin/perl
@nazwy = ("stycznia","lutego","marca","kwietnia","maja","czerwca","lipca",
 "sierpnia","września","października","listopada","grudnia");
```

```
$tekst = "Przyszedł 1-1-2001. Rozpoczął pracę 16-3-2001 i pracował do 17-11-2002.\n";
$tekst =~ s/-(\d+)-/ $nazwy[$1-1] /g;
print $tekst;
```

Na początku tworzymy tablicę `nazwy` z nazwami miesięcy w odpowiednim przypadku. Następnie do zmiennej `tekst` wstawiamy dwa zdania z trzema datami. Trzeci wiersz to właściwa zamiana.

Szukamy wzorca rozpoczynającego się minusem (`-`), następnie przynajmniej jedna cyfra (`\d+`) i potem znowu minus. Celem zamiany jest odstęp (spacja), wartość pewnej zmiennej i znowu spacja. Po dopasowaniu do wzorca pod zmienną `$1` podstawione zostanie wyrażenie pasujące do fragmentu wzorca w nawiasach. Będzie to liczba oznaczająca numer miesiąca. Ponieważ pierwszy miesiąc w tablicy `nazwy` ma numer zero, więc od zmiennej należy odjąć jeden (`$1-1`). Nazwa miesiąca pobrana z tablicy `nazwy` ma właśnie taki numer i jest używana przy zamianie. Efekt podmiany można zobaczyć na ekranie.

```
Przyszedł 1 stycznia 2001. Rozpoczął pracę 16 marca 2001 i pracował do 17 listopada 2002.
```

Dzielenie

Ćwiczenie 7.22.

Wstaw do tablicy slowa kolejne słowa z wyrażenia zawartego w zmiennej tekst.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "Tu znajduje się pięć słów";
@slowa = split(/\s+/,$tekst);
print join("\n",@slowa);
```

Wyrażenia regularne można stosować także do konwersji zmiennej skalarnej na tablicę. Służy do tego funkcja `split`, której jako pierwszy argument podajemy wyrażenie regularne, a jako drugi — zmienną, na której podział ma być dokonany. W tym przypadku poszczególne elementy tablicy `slowa` są oddzielone białymi znakami (zazwyczaj słowa oddzielamy spacjami). Pominięcie plusa, występującego w pierwszym wierszu, może spowodować, że niektóre elementy tablicy `slowa` będą puste. Stanie się tak, gdy obok siebie znajdą się co najmniej dwa białe znaki.

Przeciwne działanie do funkcji `split` ma funkcja `join`. Dzięki tej drugiej możemy tablicę połączyć w zmienną skalarną, wstawiając pomiędzy elementy podany ciąg znaków (nie wyrażenie regularne). U nas będzie to znak nowej linii, czyli kolejne elementy tablicy `slowa` zobaczymy na ekranie w kolejnych wierszach.

```
Tu
znajduje
się
pięć
słów
```

Ćwiczenie 7.23.

Wstaw do tablicy litery kolejne znaki ze zmiennej tekst.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "ABCcab";
@litery = split(//,$tekst);
print join ",",@litery;
```

W przypadku, gdy używamy funkcji `split` i podamy jej puste wyrażenie regularne, w wynikowej tablicy (`litery`) otrzymamy kolejne znaki ze zmiennej `tekst`. Elementami tablicy będą wszystkie znaki, a nie jedynie litery. W naszym przypadku na ekranie zobaczymy je oddzielone przecinkami, dzięki funkcji `join`.

```
A,B,C,c,a,b
```

Transliteracja

Ćwiczenie 7.24.

Zamień wszystkie litery typowe dla języka polskiego na ich obce odpowiedniki.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "Zżółknął śledź";
$tekst =~ tr/ąćęłńóśźżĄĆĘŁŃÓŚŹŻ/acelnoszzACELNOSZZ/;
print $tekst;
```

W Perlu istnieje operator transliteracji `tr` (zapisywany także jako `y`). Jego działanie polega na zamianie wszystkich znaków ze wzorca na odpowiadające im znaki ze wzorca zamiany. W naszym przypadku wszystkie litery `ą` zostaną zastąpione literami `a`, `ć` zamienione zostaną na `c`, `ę` na `e` itd. Na ekranie w tym przypadku zobaczymy następujący napis:

```
Zzolknal sledz
```

Ćwiczenie 7.25.

Zamień wszystkie wielkie litery na małe.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "Na kLaWIatuRZe jEsT klAwIsz SHIFT";
$tekst =~ y/A-Z/a-z/;
print $tekst;
```

Wykorzystamy tu operator transliteracji `tr`, lecz zapisany krócej (i równoważnie) jako `y`. Zestaw wszystkich wielkich liter uzyskujemy stosując konstrukcję `A-Z` i analogicznie otrzymamy zestaw małych liter. Efekt można zobaczyć na ekranie.

```
na klawiaturze jest klawisz shift
```

Ta zamiana nie uwzględnia liter typowych dla języka polskiego. Należy dla nich wykonać dodatkową operację transliteracji.

```
#!/usr/bin/perl
$tekst = "piSZĄc uŻywAł KlaWiszA SHIFT";

$tekst =~ y/A-Z/a-z/;
$tekst =~ tr/ĄĆĘŁŃÓŚŹŻ/ąćęłńóśźż/;
print $tekst;
```

Tym razem nasz tekst będzie wyglądał tak:

```
pisząc używał klawisza shift
```

Ćwiczenie 7.26.

Policz, ile razy w zmiennej tekst występuje litera a.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "abbac";
$liczba = $tekst =~ y/a//;
print "W tekście $tekst litera a wystąpiła $liczba razy\n";
```

Liczba znaków we wzorcu i specyfikacji zamiany nie musi być równa. Litery `a` w naszym przypadku nie zostaną zamienione, a na ekranie zobaczymy następującą informację:

```
W tekście abbac litera a wystąpiła 2 razy
```

Ćwiczenie 7.27.

Wyrzuć powtarzające się obok siebie znaki ze zmiennej tekst.

Rozwiązanie

```
#!/usr/bin/perl
$tekst = "abbac##";
$tekst =~ tr/a-zA-Z//s;
print $tekst;
```

Zastosowanie modyfikatora `s` (litera na końcu) dla operatora `tr` powoduje, że wszystkie takie same znaki znajdujące się obok siebie traktowane są podczas zamiany jako jeden. Dzięki zastosowaniu operacji zapisanej powyżej, zamianie (kasowaniu duplikatów) ulegną tylko litery. Pojawi się następująca wartość zmiennej:

```
abac##
```

Należy zwrócić uwagę, że usunięte zostały jedynie litery znajdujące się obok siebie. Litera `a` występowała dwukrotnie i tak jest nadal. Usuwanie wszystkich powtórzeń spowoduje zapis przedstawiony poniżej.

```
#!/usr/bin/perl
$tekst = "abbac##";
$tekst =~ tr///cs;
print $tekst;
```

We wzorcu nie ma wpisanych żadnych znaków. Jednakże dzięki modyfikatorowi `c`, oznaczającemu odwrócenie dopasowania wzorca, operator `tr` wykona zadane operacje na każdym znaku. Nie będzie to zamiana (wzorzec zamiany jest pusty), lecz usunięcie powtórzeń (modyfikator `s`). Tym razem efekt będzie inny.

```
abac##
```

Operator `tr` może w tym przypadku zostać zastąpiony przez operator zamiany `s`.

```
#!/usr/bin/perl
$tekst = "abbac##";
$tekst =~ s/(.)\1+/$1/g;
```

Widzimy tu dowolny znak oznaczony kropką ujętą w nawiasy. W ten sposób dla każdego znaku (modyfikator g) ustawiona zostaje zmienna `$1`, której wartością jest ten znak. Ustawionej zmiennej nie można jednak używać w lewej części operatora (we wzorcu). Zamiast niej używa się odpowiadającego jej wyrażenia — `\1`. Zatem sekwencja `(.)\1+` oznacza dowolny znak, a po nim ten sam znak, występujący co najmniej raz (znak `+`). Zestaw takich samych znaków zostanie zamieniony na jednokrotne wystąpienie znaku (`$1`).