

Perl. Wprowadzenie

Autorzy: Randal L. Schwartz, Tom Christiansen

Tłumaczenie: Michał Jęczalik, Marcin Jagodziński (wstęp)

ISBN: 83-7197-220-2

Format: B5, 256 stron

Zawiera CD-ROM

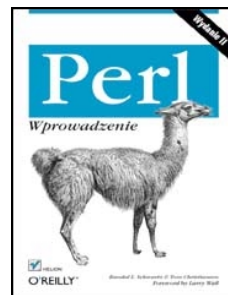
Data wydania: 07/2000

Cena książki: 36.00 zł

Przesyłka gratis! Odbiorca pokrywa jedynie koszty pobrania (aktualnie 2,90 zł) w przypadku przesyłki za zaliczeniem pocztowym

Jeżeli znasz tę książkę [ocień ją](#)






Jeżeli chciałbyś zgłosić wpis do erraty [daj nam znać](#)



W nowym wydaniu sławnego bestsellera dwaj czołowi eksperci w dziedzinie języka Perl odsłaniają przed nami tajniki najbardziej uniwersalnego spośród języków skryptowych ery WWW. Ten starannie opracowany podręcznik, poprzedzony przedmową autorstwa twórcy języka, Larryego Walla, uznawany jest za półoficjalny podręcznik szkoleniowy i praktyczny przewodnik programisty. Najnowsze, zaktualizowane wydanie książki obejmuje wersję 5.004 języka Perl.

"Perl. Wprowadzenie" to praktyczny kurs programowania zorientowany na przekazanie czytelnikowi wiedzy pozwalającej mu na jak najszybsze rozpoczęcie własnych doświadczeń. Każdy rozdział książki uzupełniony jest ćwiczeniami i odpowiedziami do nich. Nowością w tym wydaniu jest obszerny rozdział traktujący o programowaniu CGI, a także informacje o użyciu modułów bibliotecznych, referencji i konstrukcji obiektowych.

Perl jest językiem przeznaczonym do przetwarzania plików, tekstów i manipulowania procesami. Wchodzi on standardowo w skład większości systemów z rodziny Unix, a jego implementacje są dostępne bez opłat również dla pozostałych ważniejszych systemów operacyjnych.

-  **Perl. Wprowadzenie – Spis treści**
-  **Aktualny cennik książek e-mailem**
-  **Książki i „3D” Online**
-  **Informacje o nowościach**
-  **Zamów najnowszy katalog**

19

Tworzenie skryptów CGI

Jeżeli nie spędziłeś ostatnich lat samotnie na bezludnej wyspie, słyszałeś już zapewne o WWW. Właściwie każdy spotkał się już z wszechobecnymi adresami WWW (URL-ami) umieszczonymi na tablicach reklamowych, w gazetach, raportach rządowych i napisach wyświetlanych pod koniec filmów.

Większość z bardziej rozbudowanych witryn zawiera tzw. formularze, do których wprowadza się pewne dane i wysyła poprzez kliknięcie przycisku lub obrazka. Na serwerze zostaje wówczas uruchomiony program analizujący przesłane dane i tworzący nową stronę WWW. Czasem taki program (najczęściej nazywany *skryptem CGI*) jest tylko interfejsem pośredniczącym pomiędzy bazą danych a użytkownikiem – przystosowuje pozyskane dane do postaci zrozumiałej dla bazy oraz przetwarza pobrane z niej informacje do postaci zrozumiałej dla przeglądarki (czyli najczęściej tworzy kod HTML).

Skrypty CGI pozwalają na znacznie więcej niż tylko proste przetwarzanie przesłanych danych – można je także wywoływać na przykład po kliknięciu obrazka i mogą zwracać dowolne dane akceptowane przez przeglądarkę. Witryny internetowe wzbogacone o możliwość dynamicznego tworzenia ich zawartości (co umożliwiają właśnie skrypty CGI) stają się ciekawsze i bardziej interaktywne.

Pomimo zalewu animacji i wszechobecných reklam, strony WWW nadal zawierają dużą ilość tekstu. Perl potrafi przetwarzać tekst, dane binarne, pliki, a także komunikować się w sieci. Staje się więc idealnym narzędziem do programowania w sieci WWW.

W niniejszym rozdziale nie tylko poznamy podstawy tworzenia skryptów CGI, ale także przy okazji zapoznamy się z referencjami, modułami i programowaniem obiektowym. Pod koniec zostanie zamieszczony mały pokaz możliwości Perla w zakresie innych zastosowań związanych z Internetem.

Rozdział ten (podobnie, jak jakikolwiek dokument zawierający mniej niż kilkaset stron) siłą rzeczy nie zawiera dokładnego opisu wszystkich poruszanych tematów (np. programowania obiektowego lub użycia referencji). Zaprezentowane tutaj przykłady mogą się jednak stać doskonałym wprowadzeniem do tych (nieco bardziej już zaawansowanych) tematów. Osoby, które lubią dochodzić do wszystkiego metodą prób i błędów, będą mogły stworzyć zaawansowane skrypty, bazując tylko na przykładach zamieszczonych w tym rozdziale.

W rozdziale tym przyjęto założenie, że Czytelnik jest przynajmniej w stopniu podstawowym obeznany z językiem HTML.

Moduł CGI.pm

Poczynając od wersji 5.004, standardowa dystrybucja Perla zawiera wszechstronny moduł CGI.pm, napisany przez Lincolna Steinai, pozwalający tworzyć skrypty CGI w banalnie prosty sposób. Podobnie jak sam Perl, CGI.pm jest niezależny od architektury i można go użyć we właściwie dowolnym systemie – od Uniksa po VMS, a nawet Windows i MacOS.

Jeśli w systemie znajduje się także podręcznik systemowy dla modułu CGI.pm, warto go przeczytać – zawiera dużo wartościowych informacji. W innym wypadku można też przejrzeć sam plik *CGI.pm* – została w nim zawarta dokumentacja, zapisana w prostym formacie *pod*.

Podczas tworzenia skryptów CGI, nieocenioną pomocą jest podręcznik systemowy do tego modułu – zawiera on nie tylko opisy funkcji, ale również wiele wartościowych i ciekawych przykładów.

Środowisko działania programu CGI

Rysunek 19.1 przedstawia związek pomiędzy przeglądarką WWW, serwerem i skryptem CGI. Po kliknięciu odnośnika, użytkownik jest przenoszony w miejsce definiowane przez URL powiązany z tym odnośnikiem. URL określa serwer i pewien określony zasób udostępniany przez ten serwer. Przeglądarka komunikuje się więc z serwerem poprzez żądania pobrania konkretnych zasobów. Jeśli zasobem tym jest na przykład strona zawierająca formularz HTML, serwer w odpowiedzi wysłał go przeglądarce.

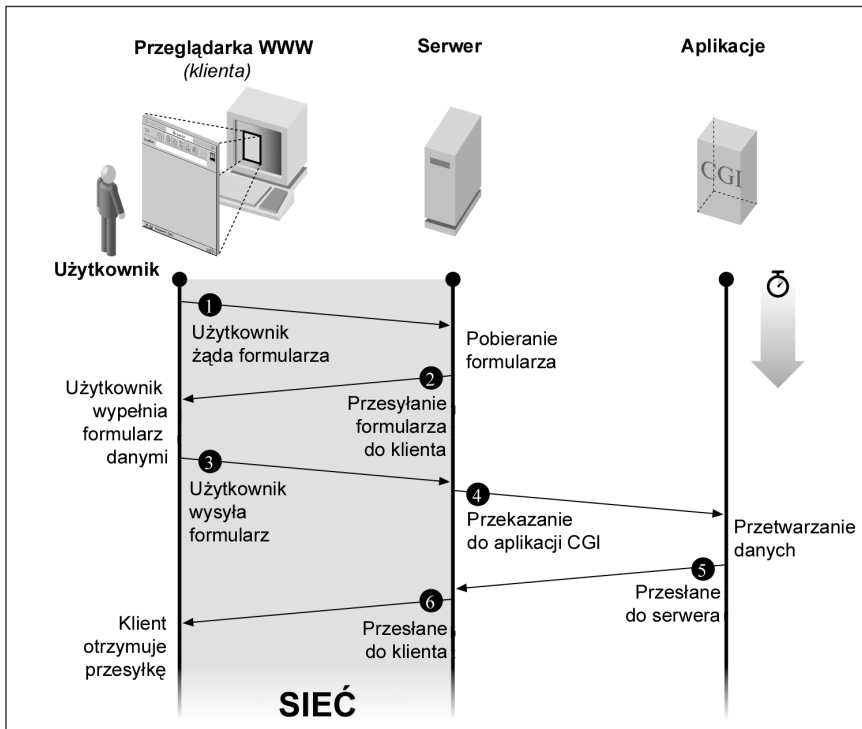
Każde pole tekstowe w formularzu posiada swoją nazwę (zapisaną w jego kodzie) i wartość (czyli to, co użytkownik w to pole wprowadzi). Formularz jest z kolei (poprzez parametr `action` znacznika `<FORM>`) przypisany do określonego skryptu CGI, który przetwarza dane. Po wypełnieniu formularza i kliknięciu przycisku „Wyślij”, przeglądarka wysyła dane do skryptu. Jest to realizowane poprzez „skok” pod URL skryptu z dołączonym tzw. *łańcuchem zapytania*, zawierającym jedną lub kilka par `nazwa=wartosc`. Każda nazwa reprezentuje jedno z pól formularza, a wartość to wpisane do niego dane. Zatem URL, pod który przeglądarka wysyła dane z formularza, wygląda mniej więcej tak (sam łańcuch zapytania to wszystko po pytajniku)¹:

http://www.COS.pl/cgi-bin/jakis_skrypt?smak=waniliowy&wielkosc=podwojne

W tym przykładzie występują dwie pary `nazwa=wartosc`, oddzielone znakiem `&`. Jest to szczegół, którym nie należy się przejmować w przypadku używania modułu CGI.pm. Część adresu `/cgi-bin/jakis_skrypt` wymaga jeszcze dalszego wyjaśnienia, lecz na tym etapie istotne jest tylko to, że jest to ścieżka do odpowiedniego skryptu CGI, przetwarzającego dane z formularza.

Kiedy serwer (czyli w tym wypadku *www.COS.pl*) odczyta przesłany przez przeglądarkę URL, uruchamia skrypt CGI i przekazuje mu jako argumenty wszystkie pary `nazwa=wartosc`. Program wykonuje swoje zadanie i (zazwyczaj) zwraca serwerowi kod HTML, który jest z powrotem odsyłany do przeglądarki.

¹ Opis ten odpowiada jednej z metod wysyłania danych z formularza: metodzie GET. W praktyce częściej używana jest inna metoda – POST, w której dane nie są przekazywane poprzez URL. Dla programisty używającego CGI.pm nie ma większego znaczenia, która metoda zostanie użyta – przyp. red.



Rysunek 19.1. Współpraca formularza z CGI.

Konwersacja między przeglądarką a serwerem (a także między serwerem a skrypcem) to tzw. protokół HTTP. Zazwyczaj nie interesuje to programisty, gdyż obsługą takich niskopoziomowych zadań zajmuje się moduł CGI.pm. Metodę wysyłania skrypcowi jego argumentów (a także innych informacji) określa specyfikacja *Common Interface Gateway* (CGI). I po raz kolejny – nie musisz w nią wnikać – zadaniami takimi jak dostarczenie do skryptu wartości argumentów zajmuje się moduł CGI.pm.

Należy także pamiętać, że skrypty CGI mogą współpracować z każdym dokumentem HTML, nie tylko z formularzami. Można na przykład napisać następujący kod:

```
Kliknij <a href="http://www.COS.pl/cgi-bin/fortune.cgi">tutaj</a>, aby
zobaczyc swoja przepowiednie.
```

fortune.cgi może być programem wywołującym po prostu dostępny na większości systemów uniksowych program *fortune*. W takim wypadku skrypcowi CGI nie są dostarczane żadne argumenty. Można też tak skonstruować dokument HTML, by dawał możliwość kliknięcia któregoś z dwóch odnośników – jeden z nich wywołuje program *fortune*, a drugi wypisuje aktualną datę. Obydwa te odnośniki mogą wskazywać na ten sam skrypt, jeśli pierwszy z nich posiada argument *fortune*, a drugi – argument *data*:

```
<a href="http://www.COS.pl/cgi-bin/fortune_lub_data?fortune">
<a href="http://www.COS.pl/cgi-bin/fortune_lub_data?data">
```

Skrypt CGI (czyli w tym przypadku *fortune_lub_data*) sprawdza, który z dwóch możliwych argumentów został przekazany i w zależności od tego, wykonuje albo program *fortune*, albo *date*.

Jak widać, argumenty nie muszą mieć postaci typowej, czyli `nazwa=data`. Skrypt CGI może wykonywać właściwie wszystko, co się programiście wymarzy, i otrzymywać dowolną liczbę argumentów.

Najprostszy skrypt CGI

Oto źródło naszego pierwszego skryptu, który jest tak prosty, że nie wymaga nawet użycia modułu CGI.pm

```
#!/usr/bin/perl -w
# hej - najprostszy skrypt CGI
print <<KONIEC_Dlugiego_Tekstu;
Content-Type: text/html

<HTML>
  <HEAD>
    <TITLE>Witaj, swiecie!</TITLE>
  </HEAD>
  <BODY>
    <H1>Pozdrowienia, Uzytkowniku!</H1>
  </BODY>
</HTML>

KONIEC_Dlugiego_Tekstu
```

Przy każdym uruchomieniu tego skryptu wyświetlany jest identyczny tekst. Nie jest to specjalnie ciekawe, ale będziemy to jeszcze rozbudowywać.

Program ten zawiera tylko jedną instrukcję – wywołanie funkcji `print`. Jej nieco dziwny argument nazywany jest *dokumentem tutaj*. Zaczyna się dwoma znakami mniejszości i słowem, które jest tzw. *znacznikiem końca*. Dla programisty shellowego taka konstrukcja może przypominać przekierowanie wejścia-wyjścia, ale to błędne wrażenie. W Perlu zapisuje się w ten sposób łańcuch zawierający kilka wierszy. Zaczyna się on w następnym wierszu i ciągnie aż do wiersza zawierającego łańcuch końcowy (który musi być umieszczony na samym początku wiersza). *Dokumenty tutaj* są bardzo przydatne podczas tworzenia kodu HTML.

Pierwsza część tego długiego łańcucha jest prawdopodobnie najważniejsza: wiersz `Content-Type` określa typ tworzonych danych. Następnie musi być umieszczony kolejny wiersz, pusty, nie zawierający żadnych spacji ani znaków tabulacji. Początkujący programiści często o nim zapominają, co jest dużym błędem, gdyż oddziela on nagłówek (analogicznie jak w przypadku nagłówka e-maila) od opcjonalnej treści². Po pustym wierszu wypisujemy kod HTML, który jest wysyłany do przeglądarki, a następnie przez nią wyświetlany.

Najpierw należy się upewnić, czy skrypt działa poprawnie, kiedy zostanie uruchomiony z wiersza poleceń. Jest to potrzebny (choć niewystarczający) krok, pozwalający upewnić się, czy został on poprawnie napisany. Istnieje wiele rzeczy, które mogą zawieść; zostały one opisane w części „*Problemy ze skryptami CGI*”.

Po pomyślnym przejściu powyższego testu, skrypt musi zostać zainstalowany na serwerze. Aby dowiedzieć się, w którym katalogu i w jaki sposób można instalować skrypty, najlepiej zapytać administratora systemu lub administratora danej witryny WWW – nie ma bowiem wypracowanych reguł.

² Nagłówek jest wymagany przez protokół HTTP, o którym już wspominaliśmy – przyp. red.

Po zainstalowaniu skryptu można go uruchomić podając jego ścieżkę jako część adresu URL. Na przykład, jeśli skrypt nazywa się *czesc*, URL może wyglądać tak: `http://www.COS.pl/cgi-bin/czesc`.

Na serwerach długie ścieżki dostępu mają zazwyczaj przypisane aliasy, dzięki czemu w powyższym przykładzie serwer przekształca `cgi-bin/czesc` w np. `usr/etc/httpd/cgi-bin/czesc`. Więcej informacji na ten temat może udzielić administrator systemu lub witryny.

Przekazywanie parametrów

Aby przekazać skryptowi parametry, zazwyczaj nie jest konieczny formularz, o czym można się przekonać zmieniając URL z poprzedniego przykładu na `http://www.COS.org/cgi-bin/lody?smak=mietowy`

Podczas odczytywania tego adresu URL przeglądarka nie tylko żąda od serwera uruchomienia skryptu *lody*, ale także przekazuje mu łańcuch `smak=mietowy`. Zadaniem skryptu jest odczytanie i właściwe zinterpretowanie tego łańcucha, co nie jest takim łatwym zadaniem, jak mogłoby się wydawać (jest on bowiem przekazywany w postaci zakodowanej). Wiele skryptów próbuje to robić we własnym zakresie, lecz nawet stosowane w nich bardzo skomplikowane algorytmy w pewnym momencie okazują się błędne lub niewystarczające. Nie warto przysparzać sobie potencjalnych kłopotów, zwłaszcza gdy istnieją gotowe moduły rozwiązujące te skomplikowane zagadnienia.

Najlepiej użyć modułu `CGI.pm`, który zawsze poprawnie przetwarza żądania CGI. Aby skorzystać z niego w swym programie, na jego początku należy umieścić taką instrukcję³:

```
use CGI;
```

Instrukcja `use` przypomina znaną z języka C dyrektywę `#include`, która podczas kompilacji odczytuje kod umieszczony w innym pliku. `use` pozwala też podać dodatkowe argumenty, określające, które funkcje i zmienne pochodzące z tego modułu mają zostać użyte. Argumenty te należy wpisać po nazwie modułu w instrukcji `use`. Później z zaimportowanych zmiennych i funkcji można korzystać tak, jakby były one umieszczone bezpośrednio w naszym programie.

Na razie z modułu `CGI.pm` potrzebujemy tylko funkcji `param()`⁴. Jeśli nie podamy jej żadnych argumentów, zwróci listę wszystkich pól, które zawierał formularz wysyłający dane do skryptu. (W tym konkretnym przypadku jest to tylko pole `smak`. Ogólnie mówiąc, jest to lista wszystkich nazw z łańcuchów `nazwa=wartosc` otrzymanych z formularza). Jeśli podamy argument będący nazwą pola, funkcja `param()` zwróci wartość (lub wartości) odpowiadającą tej nazwie. A zatem `param("smak")` zwróci `mietowy`, gdyż na końcu URL-a umieściliśmy ciąg `?smak=mietowy`.

Mimo że importujemy tylko jedną funkcję, zastosujemy notację `qw()` – ta metoda pozwala na łatwe rozszerzenie listy w przyszłości.

```
#!/usr/bin/perl -w
# cgi-bin/lody: skrypt powiadamiajacy o ulubionym smaku lodow (wersja 1)

use CGI qw(param);
print <<KONIEC;
Content-Type: text/html
```

³ Nazwy wszystkich modułów Perla mają rozszerzenie `.pm`, które instrukcja `use` automatycznie dodaje. Technika pisania modułów została opisana w rozdziale 5. książki *Perl – programowanie* i w podręczniku `perlmod(1)` – przyp. red.

⁴ Niektóre z modułów automatycznie eksportują wszystkie swoje funkcje. Ale ponieważ `CGI.pm` jest modułem obiektowym ukrywającym zwykły moduł, musimy jego funkcji zażądać oddzielnie – przyp. red.

```

<HTML>
  <HEAD>
    <TITLE>Witaj, swiecie</TITLE>
  </HEAD>
  <BODY>
    <H1>Pozdrowienia, uzytkowniku!</H1>
KONIEC

my $ulubiony = param("smak");
print "<P>Twój ulubiony smak lodow to $ulubiony.";
print <<KONIEC;
  </BODY>
</HTML>
KONIEC

```

Mniej wpisywania

Niestety, poprzedni program nadal wymaga dużo pisania. Moduł CGI.pm oferuje doskonały zestaw funkcji pozwalających uprościć wiele typowych operacji. Każda z tych funkcji zwraca łańcuch – na przykład `header()` zwraca ciąg znaków zawierający definicję `Content-type` (wraz z kolejnym pustym wierszem), `start_html(lancuch)` jako tytuł strony zwraca `lancuch`, `h1(lancuch)` zwraca `lancuch` jako nagłówek pierwszego stopnia, a `p(lancuch)` zwraca `lancuch` jako nowy akapit.

Można zapisać wszystkie te funkcje w liście funkcji do zaimportowania w instrukcji `use`, ale szybko stałaby się ona zbyt rozbudowana. CGI.pm, podobnie jak wiele innych modułów, pozwala używać etykiet, które odpowiadają różnym grupom importowanych funkcji. Wystarczy po prostu na początku listy umieścić poprzedzoną dwukropkiem nazwę etykiety. Oto spis tych dostępnych w module CGI.pm:

```

:cgi
  Importuje metody związane z przetwarzaniem argumentów, np. param().

:form
  Importuje metody tworzenia formularzy, np. textfield().

:html2
  Importuje metody tworzenia elementów HTML 2.0.

:html3
  Importuje metody tworzenia elementów HTML 3.0 (takie, jak <table>, <sup> i <sub>).

:netscape
  Importuje metody tworzenia rozszerzeń kodu HTML, stworzonych przez firmę Netscape.

:shortcuts
  Importuje metody ułatwiające tworzenie kodu HTML (czyli html2 + html3 + netscape).

:standard
  Importuje metody „standardowe”: html2, form i cgi.

:all
  Importuje wszystkie dostępne metody – ich pełna lista jest zapisana w samym kodzie modułu CGI.pm w definicji zmiennej %TAGS.

```

My będziemy używać przede wszystkim `:standard`. (Dokładniej tematyka importowania funkcji i zmiennych z modułów została opisana przy okazji omawiania modułu `Exporter` w rozdziale 7. książki *Perl – programowanie* i w dokumentacji do *Exporter(3)*).

Tak prezentuje się nasz program po użyciu dostarczanych przez `CGI.pm` skrótów:

```
#!/usr/bin/perl -w
# cgi-bin/lody: skrypt powiadamiajacy o ulubionym smaku lodow (wersja 2)

use CGI qw(:standard);
print header(), start_html("Witaj, swiecie!"), h1("Pozdrowienia, uzytkowniku!");
my $ulubiony = param("smak");
print p("Twoj ulubiony smak lodow to $ulubiony.");
print end_html();
```

Znacznie prostsze, prawda? Nie trzeba przejmować się dekodowaniem formularza, nagłówkami ani kodem HTML.

Tworzenie formularza

Zamiast wpisywać parametry skryptu ręcznie, w przeglądarce, warto stworzyć sobie formularz. Te jego części, które służą do komunikacji z użytkownikiem, to tzw. pola. Pola dostępne w formularzach to między innymi jedno- i wielowierszowe pola wprowadzania tekstu, menu, przewijane listy, przeróżne przyciski i pola zaznaczania.

Spróbujmy utworzyć stronę HTML, zawierającą formularz z jednym polem wprowadzania tekstu i przyciskiem służącym do przekazania danych serwerowi. Po jego kliknięciu⁵ jest wykonywany skrypt podany w atrybucie `ACTION`, czyli w naszym przypadku *lody*.

```
<!-- lody.html -->
<HTML>
  <HEAD>
    <TITLE>Lodziarnia</TITLE>
  </HEAD>
  <BODY>
    <H1>Lodziarnia</H1>
    <FORM ACTION="http://www.COS.pl/cgi-bin/lody">
      Jaki smak lodow lubisz? <INPUT NAME="smak" value="mietowy">
    <P>
      <INPUT TYPE="submit">
    </FORM>
  </BODY>
</HTML>
```

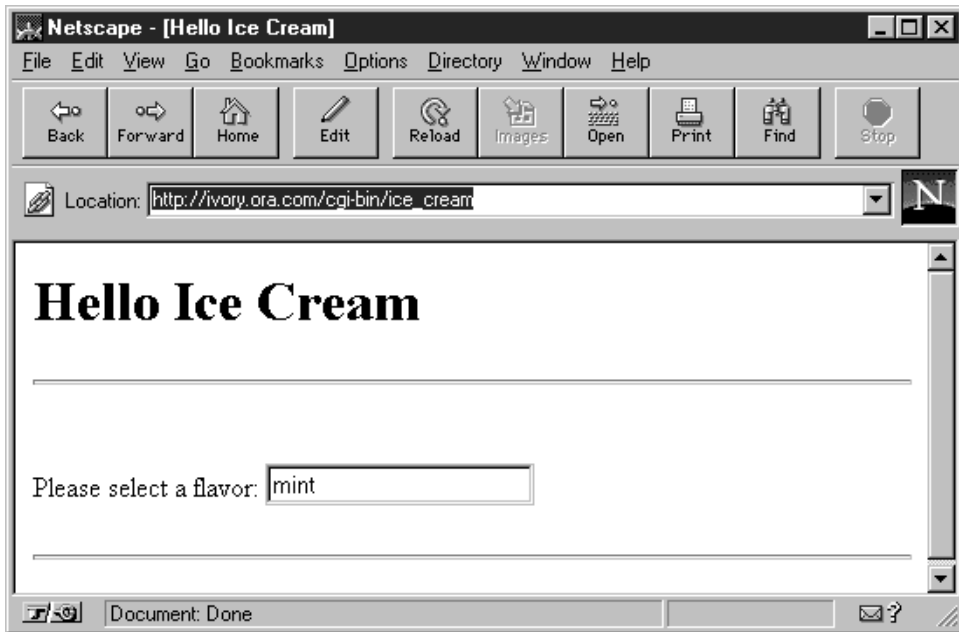
Należy pamiętać, że skrypt CGI może wygenerować dowolny kod HTML, który jest przekazywany przeglądarce. Wynika z tego, że może także stworzyć stronę zawierającą formularz. Ten sam program może jednocześnie przyjąć i przetworzyć dane z tego formularza. Wszystko, czego do tego potrzeba, to podział programu na dwie części, z których każda będzie wykonywała odmienne zadanie. Właściwa część zostanie wybrana na podstawie przekazanych skryptowi parametrów. Jeśli nie otrzyma on żadnych argumentów, wyśle przeglądarce formularz; w innym przypadku parametry zawierają dane wprowadzone do poprzednio wysłanego formularza, na podstawie których skrypt tworzy odpowiedź.

⁵ Niektóre przeglądarki pozwalają nie umieszczać przycisku wysyłania w przypadku, gdy formularz ma tylko jedno pole wprowadzania tekstu. Po jego wypełnieniu wystarczy nacisnąć klawisz Enter. Jednak zawsze najlepiej jest używać przenośnego kodu HTML – `przyp. red`.

Wykonywanie wielu zadań przez jeden skrypt pozwala na łatwą nad nimi kontrolę, jedynym minusem jest nieco dłuższy czas uruchamiania go podczas wczytywania strony z formularzem. Oto kod:

```
#!/usr/bin/perl -w
# cgi-bin/lody: skrypt powiadamiający o ulubionym #smaku lodow (wersja 3)
use CGI qw(:standard);
my $ulubiony = param("smak");
print header, start_html("Lodziarnia"), hl("Lodziarnia");
if ($ulubiony) {
    print q("Twój ulubiony smak lodow to $ulubiony.");
} else {
    print hr, start_form; # hr() tworzy pozioma #kreske: <HR>
    print q("Wybierz smak: ", textfield("smak","mietowy"));
    print end_form, hr;
}
```

Jeśli użytkownik kliknie w przeglądarce odnośnik wskazujący na ten skrypt (jeśli nie podaje on parametrów w formie ?cokolwiek na końcu URL-a), zobaczy to, co na rysunku 19.2. Pole tekstowe zawiera domyślną wartość, ale nic nie stoi na przeszkodzie, aby ją zmienić.



Rysunek 19.2. Przykład prostego formularza wypełnionego danymi.

Teraz wystarczy wypełnić pole `smak`, nacisnąć klawisz `Enter`. Efekt będzie mniej więcej taki, jak na rysunku 19.3.



Rysunek 19.3. Rezultat wysłania formularza z rysunku 19.2.

Inne elementy formularzy

Wiemy już, w jaki sposób tworzyć proste pola tekstowe w formularzu i odpowiadać na dostarczone przez nie dane, ale warto jeszcze poznać sposób, w jaki tworzy się inne elementy, na przykład przyciski, pola zaznaczania i menu.

Oto nieco bardziej rozbudowana wersja naszego programu. Dodaliśmy kilka nowych elementów, takich jak menu, przycisk wysłania (o nazwie Zamów) i przycisk resetujący cały formularz (czyli usuwający wszystko, co użytkownik wprowadził). Menu zachowują się prawie tak, jak prawdziwe menu, ale argumenty funkcji `popup_menu` mogą wydawać się na razie trochę tajemnicze – do czasu przeczytania kolejnej części o nazwie „Referencje”.

Funkcja `textfield()` tworzy pole wprowadzania tekstu o określonej nazwie. Więcej szczegółów o niej zostanie podanych podczas omawiania „książki gości”.

```
#!/usr/bin/perl -w
# cgi-bin/lody: skrypt powiadamiający o ulubionym smaku lodow (wersja 4)
use strict; # wymus deklaratcje zmiennych poprzez użycie cudzysłowow
use CGI qw(:standard);
my $ulubiony = param("smak");

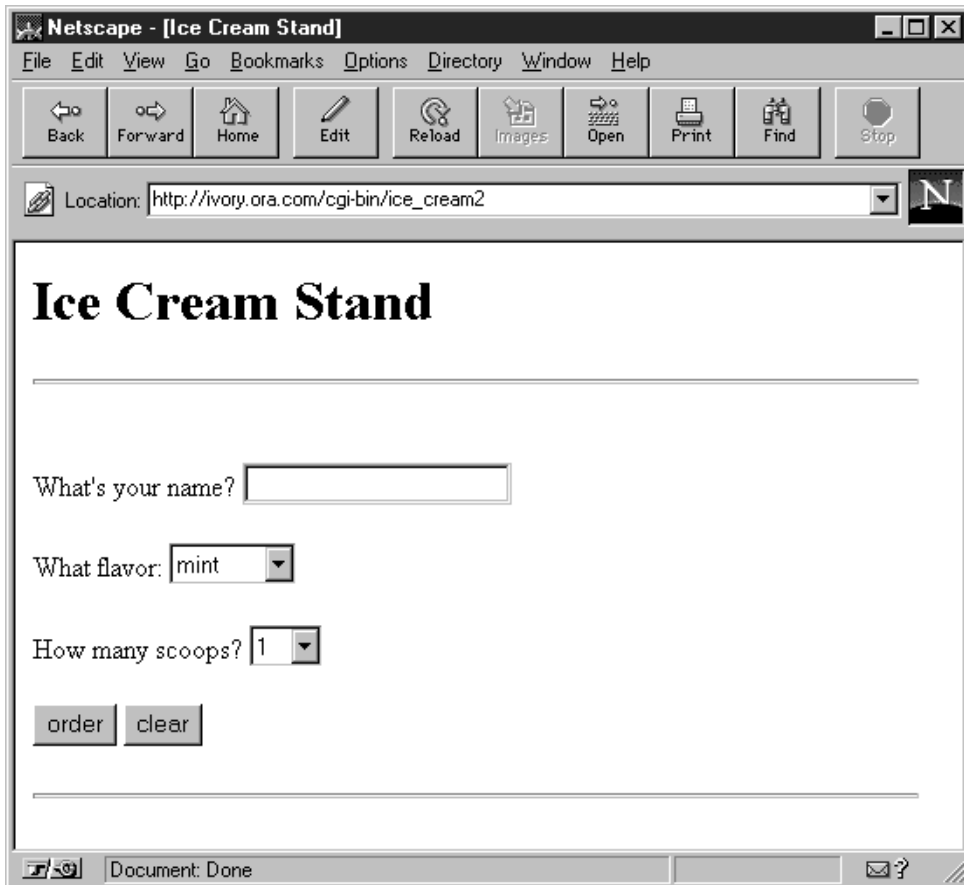
print header, start_html("Lodziarnia"), hl("Lodziarnia");
if (param()) { # formularz zostal juz wypelniony
    my $kto = param("imie");
    my $smak = param("smak");
    my $galki = param("galki");
    my $podatek = 1.22;
    my $scena = sprintf("%.2f", $podatek * (1.00 + $galki*0.25));
    print p("OK, $imie, prosze $galki galki o smaku $smak za $scena.");
}
```

```

} else { # to pierwszy raz, wiec wypisz formularz
  print hr(); # narysuj przed formularzem pozioma #kreske
  print start_form();
  print p("Jak Ci na imie? ", textfield("imie"));
  # WYJASNIENIE PONIZSZYCH DWOCH WIERSZY MOZNA ZNALEZC W NASTEPNEJ CZESCI
  print p("Jaki smak: ", popup_menu("smak", ['mietowy', 'wisniowy',
    'waniliowy']));
  print p("Ile galek? ", popup_menu("galki", [ 1..3 ]));
  print p(submit("zamow"), reset("wyczysc"));
  print end_form(), hr();
}
print end_html;

```

Rysunek 19.4 pokazuje stronę, którą widzimy po uruchomieniu skryptu.



Rysunek 19.4. Formularz nieco bardziej skomplikowany.

Funkcja `param()`, jeśli zostanie wywołana bez argumentów, zwraca nazwy wszystkich wypełnionych pól formularza. Można dzięki temu określić, czy skrypt został wywołany z formularza. Jeśli skryptowi przekazano parametry, to użytkownik wypełnił część z pól i tworzona jest odpowiedź. W innym przypadku skrypt tworzy nowy formularz.

Referencje

Nietrudno zauważyć, że funkcje `popup_menu()` w poprzednim przykładzie mają dziwne argumenty. Co to jest `['mietowy', 'wisniowy', 'waniliowy']` i `[1..3]`? Nawiasy kwadratowe tworzą coś, o czym wcześniej nie mówiliśmy – referencję do anonimowej tablicy. To wszystko dlatego, że funkcja `popup_menu` oczekuje argumentu w postaci referencji do tablicy. Innym sposobem utworzenia odniesienia do tablicy jest użycie lewego ukośnika (`\`) przed nazwą tablicy, np. `\@mozliwosci`.

A więc

```
@mozliwosci = ('mietowy', 'wisniowy', 'waniliowy');
print p("Jaki smak: ", popup_menu("smak", \@mozliwosci));
```

działa tak samo, jak

```
print p("Jaki smak: ", popup_menu("smak", ['mietowy', 'wisniowy', 'waniliowy']));
```

Referencje pełnią rolę znanych w innych językach programowania wskaźników, lecz stwarzają mniej kłopotów. Są wartościami, które odnoszą się do innych wartości (lub zmiennych). Perl używa tak zwanych „mocnych referencji”. Nie zdarza się, by powodowały one wyjście z programu z zapisem pamięci (*core dump*). Jest nawet lepiej – część pamięci, na którą wskazuje referencja, jest automatycznie zwalniana, kiedy referencja nie jest już używana. Referencje odgrywają ważną rolę w programowaniu zorientowanym obiektowo. Są także wykorzystywane w programowaniu tradycyjnym, jako podstawa struktur danych nieco bardziej skomplikowanych niż proste jednowymiarowe tablice i tablice asocjacyjne. Perl pozwala tworzyć referencje do zarówno nazwanych, jak i nienazwanych skalarów, tablic, tablic asocjacyjnych i funkcji. W taki sam sposób, w jaki można utworzyć referencję do nazwanej tablicy za pomocą `\@tablica` i do anonimowej tablicy za pomocą `[lista]`, można tworzyć referencje do normalnych tablic asocjacyjnych za pomocą `\%tablica_asocjacyjna` oraz do anonimowych⁶:

```
{ klucz1, wartosc1, klucz2, wartosc2, ... }
```

Więcej o referencjach można się dowiedzieć z rozdziału 4. książki *Perl – programowanie* lub z podręcznika *perlref(1)*.

Bardziej skomplikowane wywołania

Zakończyliśmy omawianie elementów formularza utworzeniem naprawdę skomplikowanego pola, pozwalającego użytkownikowi wybrać dowolną liczbę pozycji. Funkcja `scrolling_list()` modułu `CGI.pm` pobiera dowolną liczbę argumentów, z których każdy zawiera nazwany parametr (zaczynający się myślnikiem) i przypisaną mu wartość.

Aby dodać przewijaną listę do formularza, należy wykonać coś takiego:

```
print scrolling_list(
  -NAME => "smaki",
  -VALUES => [ qw(mietowy czekoladowy wisniowy waniliowy sliwkowy) ],
  -LABELS => {
    mietowy => "Silnie mietowy",
    czekoladowy => "Czekoladowo-czekoladowy",
    wisniowy => "Wisienka",
```

⁶ Tak, nawiasy klamrowe mają w Perlu kilka znaczeń, w zależności od kontekstu, w którym zostaną użyte – przyp. red.

```

        waniliowy => "Waniliowy",
        sliwkowy => "Idealnie sliwkowy"
    },
    -SIZE => 3,
    -MULTIPLE => 1, # 1 - prawda, 0 - fałsz
);

```

Wartości parametrów mają takie znaczenie:

-MAIN

Nazwa elementu – można jej później użyć, aby odebrać dane z formularza za pomocą funkcji `param()`.

-LABELS

Referencja do anonimowej tablicy asocjacyjnej. Jego wartości oznaczają etykiety (pozycje listy) widziane przez użytkownika. Kiedy użytkownik wybiera jedną z etykiet, do skryptu CGI zwracany jest odpowiedni klucz tablicy. Czyli jeśli użytkownik wybierze pozycję o nazwie `Idealnie sliwkowy`, program CGI otrzyma argument `sliwkowy`.

-VALUES

Referencja do anonimowej tablicy, która zawiera klucze tablicy asocjacyjnej `-LABELS`

-SIZE

Liczba oznaczająca, ile pozycji będzie widocznych dla użytkownika jednocześnie.

-MULTIPLE

Wartość *prawda* albo *fałsz* (w rozumieniu Perla), oznaczająca, czy użytkownik będzie mógł wybrać więcej niż jedną pozycję z listy naraz.

Jeśli ustawimy `-MULTIPLE` na wartość *prawda*, listę zwracaną przez funkcję `param()` trzeba będzie przypisać do tablicy:

```
@mozliwosci = param("smaki");
```

Przekazanie referencji do istniejącej tablicy asocjacyjnej to jeszcze jeden sposób utworzenia tego samego pola formularza, zamiast tworzenia go „w locie”:

```

%smaki = (
    mietowy => "Silnie mietowy",
    czekoladowy => "Czekoladowo-czekoladowy",
    wisniowy => "Wisienka",
    waniliowy => "Waniliowy",
    sliwkowy => "Idealnie sliwkowy",
);
print scrolling_list(
    -NAME => "smaki",
    -LABELS => \%smaki
    -VALUES => [ keys %smaki ],
    -SIZE => 3,
    -MULTIPLE => 1,
);

```

Tym razem zapisaliśmy wartości powstałe z kluczy tablicy asocjacyjnej `%smaki`, która sama w sobie jest przekazywana przez referencję za pomocą operatora `\` (lewy ukośnik). Warto zauważyć, że parametr `-VALUES` jest nadal ujęty w nawiasy kwadratowe. Nie istnieje możliwość przekazania po prostu wyniku funkcji `keys` jako listy, ponieważ konwencja wywoływania funkcji `scrolling_list()` wymaga w tym miejscu referencji do tablicy, co nawiasy kwadratowe zapewniają.

Nawiasy kwadratowe można traktować jako wygodny sposób zapisu kilku wartości jako jednej.

Tworzenie książki gości

Śledząc uważnie powyższe przykłady, można napisać prosty skrypt CGI. Ale co z tymi nieco trudniejszymi? Typowym zadaniem jest skrypt CGI obsługujący książkę gości, aby odwiedzający witrynę mogli umieścić tam swoje komentarze lub wiadomości.

Tak się akurat składa, że formularz dla tego skryptu jest dość prosty i znacznie mniej skomplikowany niż niektóre stosowane w przykładzie z lodami. Inne sprawy stają się nieco trudniejsze, ale dojdziemy do wszystkiego po kolei.

Zależy nam, aby wiadomości zapisywane do książki gości pozostawały tam nieco dłużej niż tylko na czas pobytu użytkownika na stronie. Musimy w tym celu mieć jakiś plik, gdzie będziemy mogli je przechowywać. Skrypt CGI działa zazwyczaj z UID innego użytkownika, więc raczej nie będzie możliwości korzystania z pliku umieszczonego w naszym katalogu macierzystym. Musimy zatem najpierw utworzyć jakiś plik dostępny dla wszystkich. W systemach uniksowych można go umieścić np. w katalogu `/tmp`:

```
touch /tmp/archiwum_ksiazki_gosci
chmod 0666 /tmp/archiwum_ksiazki_gosci
```

No dobrze, a co się stanie, jeśli kilka osób będzie korzystało ze skryptu jednocześnie? System operacyjny nie zabrania równoczesnego dostępu do plików, więc jeśli nie zachowamy pewnych środków ostrożności, plik z archiwum może ulec uszkodzeniu w chwili, gdy kilka osób zapisze go jednocześnie. Aby tego uniknąć, użyjemy funkcji Perla `flock`, pozwalającej uzyskać wyłączność w dostępie do pliku:

```
use Fcntl qw(:flock); # importuje LOCK_EX, LOCK_SH, LOCK_NB
...
flock(UCHWYT, LOCK_EX) || zakoncz("nie moze zablokowac $ARCHIWUM: $!");
```

Argument `LOCK_EX` funkcji `flock` nakazuje oczekiwanie na dostęp wyłączny⁷.

Użycie funkcji `flock` to prosta, lecz pewna metoda zablokowania pliku, nawet jeśli jej implementacje różnią się w poszczególnych systemach operacyjnych. Pozwala ona naprawdę „zawłaszczyć” plik, ponieważ kończy swoje działanie dopiero wtedy, gdy uzyska blokadę. Należy pamiętać, że blokowanie pliku ma sens tylko wtedy, gdy wszystkie programy usiłujące uzyskać do niego dostęp respektują blokadę utworzoną w ten sam sposób.

Perl i programowanie zorientowane obiektowo

Przyszedł teraz czas, by dowiedzieć się, w jaki sposób można używać obiektów i klas. Mimo że stworzenie własnego modułu obiektowego nie leży w zakresie tej książki, powinniśmy wiedzieć, w jaki sposób używać już istniejących modułów obiektowych. Bardziej szczegółowe informacje o używaniu i tworzeniu modułów można znaleźć w rozdziale 5. książki *Perl – programowanie* i w podręczniku *perltoot(1)*.

Nie będziemy się tu zagłębiać w kwestie teoretyczne. Obiekty można po prostu traktować jako pakiety (którymi zresztą są!) zawierające różne tajemnicze rzeczy, których nie wywołuje się bezpośrednio. Obiekty dostarczają podprogramów zapewniających wykonanie wszelkich koniecznych operacji.

⁷ W wersjach Perla wcześniejszych niż 5.004 wiersz `use Fcntl` należy oznaczyć jako komentarz i jako argumentu funkcji `flock` użyć po prostu liczby 2 – przyp. red.

Załóżmy, że mamy moduł CGI.pm, który zwraca obiekt o nazwie `$query`, reprezentujący dane wprowadzone przez użytkownika. Aby pobrać z niego parametr, musimy wywołać funkcję `param()`:

```
$query->param("odpowiedz");
```

Oznacza to „wykonaj funkcję `param()` z obiektu `$query`, z argumentem `odpowiedz`”. Jedyne, co różni się tutaj od wywołania zwykłej funkcji, to dodana nazwa obiektu.

Funkcje powiązane z obiektami to tzw. *metody*. Aby pobrać wartość zwracaną przez funkcję `param()`, można użyć zwykłego przypisania i zapisać ją w zwykłej zmiennej o nazwie np. `$odpowiedz`:

```
$odpowiedz = $query->param("odpowiedz");
```

Obiekty wyglądają tak, jak skalary – można je przechowywać w zmiennych skalarnych (np. w `$query` w naszym przykładzie), a także tworzyć tablice i tablice asocjacyjne obiektów. Obiektów nie można jednak traktować jak łańcuchów lub liczb. Są one specjalnym rodzajem referencji⁸, traktowanych w zupełnie inny sposób. Można przyjąć, że obiekty to specjalny, zdefiniowany przez użytkownika typ danych.

Typ danego obiektu to jego klasa. Nazwa klasy jest po prostu nazwą modułu, bez przyrostka *.pm*, i zazwyczaj słowa „klasa” i „moduł” są używane wymiennie. Możemy zatem mówić „moduł CGI” oraz „klasa CGI”. Obiekty w konkretnej klasie są tworzone i zarządzane przez moduł, który tę klasę implementuje.

Dostęp do klasy jest możliwy poprzez załadowanie do klasy modułu, który wygląda tak, jak każdy inny moduł, poza tym, że te zorientowane obiektowo zazwyczaj niczego nie eksportują. Można się posłużyć porównaniem, że klasa to fabryka, wytwarzająca gotowe obiekty. Aby nakazać klasie utworzenie jednego z nich, należy wywołać specjalne metody zwane konstruktorami. Na przykład:

```
$query = CGI->new(); # wykonaj metode new() z klasy # "CGI"
```

Powyższe to wywołanie metody klasy. Jedyne, co odróżnia ją od metody obiektu, to fakt, że przy jej wywołaniu poprzedza się ją nazwą *klasy*, a nie obiektu. Metoda obiektowa mówi „wywołaj funkcję o danej nazwie, powiązaną z tym obiektem”, a metoda klasy – „wywołaj funkcję o danej nazwie, powiązaną z tą klasą”.

Czasami można zobaczyć tę samą instrukcję zapisaną w trochę innej formie:

```
$query = new CGI; # to samo
```

Funkcjonalnie te formy są równoważne. Druga wymaga wpisywania mniejszej liczby znaków przestankowych, więc czasami bywa wygodniejsza. Trudniej jej natomiast użyć jako części większego wyrażenia.

Z punktu widzenia projektanta modułów obiektowych, obiekt jest referencją do zdefiniowanej przez użytkownika struktury danych, zazwyczaj anonimowej tablicy asocjacyjnej. Wewnątrz tej struktury są przechowywane wszystkie rodzaje informacji. Użytkownik obiektu powinien pobierać te informacje (aby je odczytać lub zmienić), nie traktując obiektu jako referencji (manipulując danymi bezpośrednio), lecz jedynie stosując metody. Zmiana danych obiektu w inny sposób może być bardzo groźna w skutkach. Aby zapoznać się z metodami i sposobem ich działania, wystarczy przeczytać dokumentację do modułu, zazwyczaj dostarczaną razem z nim.

⁸ tzw. *kwalifikowane odwołanie* – przyp. red.

Obiekty w CGI.pm

Moduł CGI jest niezwykle w tym sensie, że może być traktowany zarówno jako moduł tradycyjny, z funkcjami eksportowanymi, jak i moduł obiektowy. Niektóre z programów łatwiej napisać korzystając z interfejsu obiektowego, zamiast proceduralnego. Książka gości należy do tej pierwszej grupy. Dane wprowadzone przez użytkownika pobieramy za pomocą obiektu CGI i jednocześnie tworzymy kod HTML za pomocą tego samego obiektu.

Najpierw musimy jednak utworzyć ten obiekt. W CGI.pm, tak jak w wielu innych klasach, metodą tworzącą obiekty jest metoda klasy o nazwie `new()`⁹.

Metoda ta tworzy i zwraca nowy obiekt CGI odpowiadający wypełnionemu formularzowi. Wywołana bez żadnych argumentów metoda `new()` tworzy obiekt odczytując dane przekazane przez przeglądarkę. Sam skrypt objaśnimy za chwilę, na razie założymy, że ma on nazwę `ksiazka_gosci` i jest umieszczony w katalogu `cgi-bin`. Wygląda trochę inaczej niż omawiane wcześniej dwuczęściowe skrypty (w których jedna część zwracała formularz, a inna odczytywała z niego dane) – ten obsługuje obydwie te funkcje jednocześnie. Dlaczego? Dlatego, że nie ma w tym przypadku potrzeby tworzenia oddzielnego dokumentu HTML, zawierającego formularz zapisu. Użytkownik może uruchomić skrypt klikając po prostu taki odnośnik:

```
Wpisz sie do naszej  
<A HREF="http://www.GDZIES.pl/cgi-bin/ksiazka_gosci">ksiazki_gosci</A>
```

Skrypt zwraca wtedy przeglądarce formularz oraz wcześniej zapisane wiadomości (do określonego limitu). Użytkownik może wypełnić formularz i wysłać. Wiadomość jest dodawana do archiwum i razem z innymi ponownie zwracana, łącznie z nowym formularzem.

Użytkownik może kontynuować odczytywanie aktualnego zestawu wiadomości i wysyłanie nowych tak długo, jak mu się żywnie podoba.

Oto wspomniany skrypt. Warto rzucić na niego okiem jeszcze przed szczegółowym omówieniem jego działania.

```
#!/usr/bin/perl -w

use 5.004;
use strict; # wymus deklarowanie zmiennych przy uzyciu cudzyslowow
use CGI qw(:standard); # importuj skrotly
use Fcntl qw(:flock); # importuje LOCK_EX, LOCK_SH i #LOCK_NB

sub zakoncz { # funkcja pozwalajaca w elegancki sposob #reagowac na bledy
    my $blad = "@_";
    print hl("BLAD"), p($blad), end_html;
    die $blad;
}

my(
    $PLIK,      # nazwa pliku z archiwum
    $NAJWYZEJ, # ile pozycji przechowywac
    $TYTUL,    # tytul strony i naglowek
    $aktualne, # nowa pozycja w ksiazce
    @pozycje,  # wszystkie wpisy
    $pozycja,  # jedna, konkretna pozycja
);
```

⁹ W odróżnieniu od C++, Perl uznaje konstruktory o dowolnych nazwach, na przykład `dalej()` lub `kazek()`. Ale większość klas i tak używa konstruktora `new()` – przyp. red.

```

$TYTUL = "Prosta ksiazka gosci";
$PLIK = "/tmp/archiwum_ksiazki_gosci"; # lub #gdziekolwiek
$NAJWYZEJ = 10;

print header, start_html($TYTUL), h1($TYTUL);

$aktualne = CGI->new(); # biezace zgloszenie
if ($aktualne->param("wiadomosc")) { # w porzadku, mamy nowy wpis
    $aktualne->param("data", scalar localtime); # zapisz date i godzine
    @pozycje = ($aktualne);
}

# otworz plik w trybie odczyt-zapis (zachowujac jego zawartosc)
open(ARCHIWUM, "+<$PLIK") || zakoncz("nie moze otworzyc $PLIK: $!");

# zablokuj plik
flock(ARCHIWUM, LOCK_EX) || zakoncz("nie moze zablokowac $PLIK: $!");

# wczytaj $NAJWYZEJ pozycji, z najnowsza na pierwszym miejscu
while (!eof(UCHWYT) && @pozycje < $NAJWYZEJ) {
    $pozycja = CGI->new(\*ARCHIWUM); # przekaz uchwyt pliku jako referencje
    push @pozycje, $pozycja;
}
seek(ARCHIWUM, 0, 0) || zakoncz("nie moze przesunac sie do poczatku $PLIK: $!");
foreach $pozycja (@pozycje) {
    $pozycja->save(\*ARCHIWUM); # przekaz uchwyt pliku jako referencje
}
truncate(ARCHIWUM, tell(ARCHIWUM)) || zakoncz("nie moze obciac pliku $PLIK: $!");
close(ARCHIWUM) || zakoncz("nie moze zamknac $PLIK: $!");

print hr, start_form; # hr() tworzy pozioma kreske <HR>
print p("Imie:", $aktualne->textfield(
    -NAME => "imie"));
print p("Wiadomosc:", $aktualne->textfield(
    -NAME => "wiadomosc",
    -OVERRIDE => 1, # usuwa poprzednia wiadomosc
    -SIZE => 50));
print p(submit("wyslij"), reset("wyczysc"));
print end_form, hr;

print h2("Wczesniejsze wpisy");
foreach $pozycja (@pozycje) {
    printf("%s [%s]: %s",
        $pozycja->param("data"),
        $pozycja->param("imie"),
        $pozycja->param("wiadomosc"));
    print br();
}
print end_html;

```

Rysunek 19.5 pokazuje przykładowy zrzut ekranu po uruchomieniu skryptu. Należy zauważyć, że skrypt zaczyna się dyrektywą:

```
use 5.004;
```

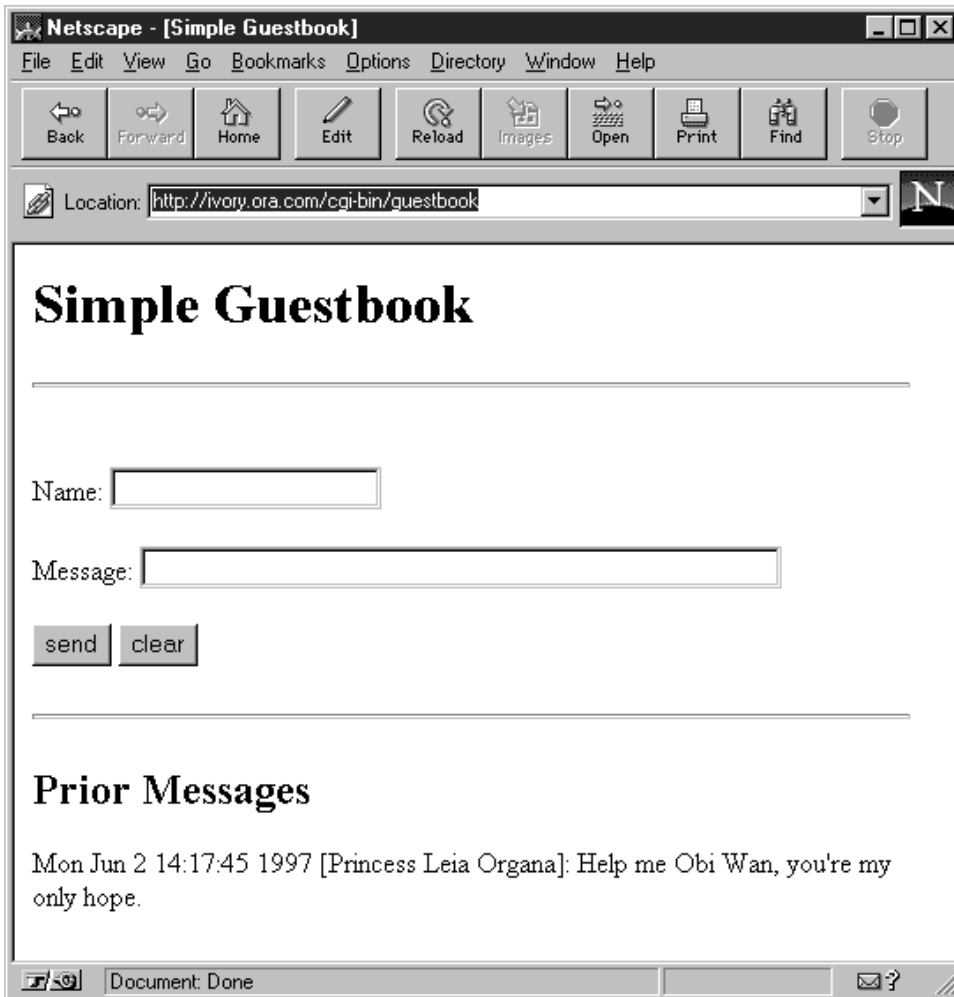
Chcąc uruchamiać skrypt z wcześniejszymi wersjami Perla niż 5, należy oznaczyć jako komentarz wiersz:

```
use Fcntl qw(:flock);
```

i zastąpić `LOCK_EX` w pierwszym wywołaniu funkcji `flock` liczbą 2.

Ponieważ każde wykonanie skryptu zwraca przeglądarce formularz, zaczynamy od wypisania kodu HTML:

```
print header, start_html($TYTUL), h1($TYTUL);
```



Rysunek 19.5. Formularz umożliwiający dokonywanie wpisów w księdze gości.

Następnie skrypt tworzy nowy obiekt CGI:

```

$aktualne = CGI->new(); # biezace zgloszenie
if ($aktualne->param("wiadomosc")) { # w porzadku, mamy #nowy wpis
    $aktualne->param("data", scalar localtime); # #zapisz date i godzine
    @pozycje = ($aktualne);
}

```

Jeśli skrypt jest wywołany przez wysłanie formularza, obiekt `$aktualne` zawiera informacje o danych wprowadzonych przez użytkownika. Formularz (zobacz poniżej) ma dwa pola: pole nazwy, zawierające nazwisko użytkownika, oraz pole wiadomości, zawierające samą wiadomość. Dodatkowo powyższy kod umieszcza datę i godzinę otrzymania danych. Przekazanie metodzie `param()` dwóch argumentów powoduje ustawienie parametru o nazwie podanej jako pierwszy argument na wartość podaną jako drugi argument. Jeśli skrypt nie został wywołany przez wysłanie formularza, tylko przez kliknięcie odnośnika „zapisz się do naszej książki gości”, to utworzony obiekt będzie pusty. Test `if` zwróci wartość `false` i do tablicy `@pozycje` nie zostanie nic dodane. W innym przypadku z pliku archiwum odczytywane są wszelkie wcześniejsze zapisy. Trafiają one do tablicy

@pozycje. Jeśli skrypt został wywołany przez wysłanie formularza, jego zawartość zostanie umieszczona jako pierwszy element tej tablicy. Najpierw jednak należy otworzyć plik z archiwum:

```
open(ARCHIWUM, "+<$PLIK") || zakoncz("nie moze otworzyc $PLIK: $!");
```

Otwiera to plik w trybie niedestruktywnego zapisu-odczytu. Możemy także użyć funkcji `sysopen()`. W taki sposób można otworzyć stary plik (jeśli taki istnieje) bez jego obcinania lub (w innym razie) utworzyć nowy:

```
# musimy zaimportowac dwie stale z modulu Fcntl
use Fcntl qw(O_RDWR O_CREAT);
sysopen(ARCHIWUM, $PLIK, O_RDWR|O_CREAT, 0666) || zakoncz("nie moze otworzyc
$PLIK: $!");
```

Później następuje zablokowanie pliku i odczytanie \$NAJWYZEJ pozycji do tablicy @pozycje:

```
flock(ARCHIWUM, LOCK_EX) || zakoncz("nie moze zablokowac $PLIK: $!");
while (!eof(UCHWYT) && @pozycje < $NAJWYZEJ) {
    $pozycja = CGI->new(\*ARCHIWUM); # przekaz uchwyt pliku jako referencje
    push @pozycje, $pozycja;
}
```

`eof` to wbudowana funkcja Perla, pozwalająca określić, czy dotarliśmy do końca pliku. Przekazując metodzie `new()` za każdym razem referencję do uchwytu pliku¹⁰, odczytujemy zachowane wcześniej pozycje, za każdym razem jedną. Następnie plik zostaje zaktualizowany w taki sposób, aby zawierał ewentualną nową pozycję:

```
seek(ARCHIWUM, 0, 0) || zakoncz("nie moze przesunac sie do poczatku $PLIK: $!");
foreach $pozycja (@pozycje) {
    $pozycja->save(\*ARCHIWUM); # przekaz uchwyt pliku #jako referencje
}
truncate(ARCHIWUM, tell(ARCHIWUM)) || zakoncz("nie #moge obciac pliku $PLIK: $!");
close(ARCHIWUM) || zakoncz("nie moze zamknac $PLIK: $!");
```

`seek`, `truncate` i `tell` to wbudowane funkcje Perla, których opisy można znaleźć w dokumentacji języka. Funkcja `seek` zmienia w tym przypadku pozycję wskaźnika, ustawiając go na początku pliku, `truncate` obcina ten plik do określonej długości, a `tell` zwraca aktualną pozycję wskaźnika pliku względem jego początku. Końcowym wynikiem tych wszystkich operacji jest zapisanie najwyżej \$NAJWYZEJ ostatnich pozycji, z najbardziej aktualną na początku.

Metoda `save()` obsługuje zapisywanie pozycji. Może ona być wywołana jako `$pozycja->save`, gdyż `$pozycja` to obiekt CGI stworzony za pomocą `CGI->new()`. Format pliku z archiwum wygląda mniej więcej tak, jak poniżej, przy czym pozycje są oddzielane pojedynczym znakiem równości:

```
NAZWA1=WARTOSC1
NAZWA2=WARTOSC2
NAZWA3=WARTOSC3
=
```

Teraz musimy zwrócić przeglądarce czysty formularz (to będzie oczywiście pierwszy formularz, który użytkownik widzi, jeśli kliknął odnośnik Zapisz się do naszej książki gości). Najpierw trochę przygotowań:

```
print hr, start_form; # hr() tworzy pozioma kreska #<HR>
```

¹⁰ Precyzując, jest to referencja do globa, ale w tym przypadku nie ma to znaczenia – przyp. red.

Jak już wspomnieliśmy, CGI.pm pozwala na użycie zarówno bezpośrednich wywołań funkcji, jak również wywołań metod przez obiekt CGI. Tutaj, dla uproszczenia kodu używamy prostych wywołań, ale na przykład w celu utworzenia pól formularza, powracamy do obiektów:

```
print p("Imie:", $aktualne->textfield(
    -NAME => "imie"));
print p("Wiadomosc:", $aktualne->textfield(
    -NAME => "wiadomosc",
    -OVERRIDE => 1, # usuwa poprzednia wiadomosc
    -SIZE => 50));
print p(submit("wyslij"), reset("wyczysc"));
print end_form, hr;
```

Metoda `textfield()` zwraca pole wprowadzania tekstu. Pierwsze z dwóch wywołań tworzy kod HTML dla pola z atrybutem `name="imie"`, a drugie tworzy pole z atrybutem `name="wiadomosc"`. Elementy formularza tworzone przez CGI.pm zachowują ich wartości pomiędzy wywołaniami (ale tylko w ciągu jednej sesji, czyli od chwili, kiedy użytkownik kliknie odnośnik *Zapisz się do naszej książki gości*). Oznacza to, że pole `name="imie"`, utworzone przez pierwsze wywołanie funkcji `textfield()`, będzie zawierało wartość imienia użytkownika, jeśli przynajmniej raz w czasie tej sesji wypełnił i wysłał formularz. A więc tworzone pole wprowadzania ma takie atrybuty:

```
NAME="imie" VALUE="Kazek Kowalski"
```

Drugie wywołanie `textfield()` jest trochę inne – nie chcemy, aby pole wiadomości zawierało tę starą wiadomość, a zatem para argumentów `-OVERRIDE=>1` nakazuje usunąć poprzednią wartość pola tekstowego i przywrócić domyślną. `-SIZE=>50` określa rozmiar wyświetlonego pola (w znakach). Inne opcjonalne argumenty poza tymi wymienionymi to `-DEFAULT=>wartosc_poczatkowa` i `-MAXLENGTH=>n`, gdzie `n` to maksymalna liczba znaków, które pole zaakceptuje.

Na koniec zwracamy wcześniej zapisane wiadomości, razem z tą, która została właśnie dodana:

```
print h2("Wczesniejsze wpisy");
foreach $pozycja (@pozycje) {
    printf("%s [%s]: %s",
        $pozycja->param("data"),
        $pozycja->param("imie"),
        $pozycja->param("wiadomosc"));
    print br();
}
print end_html;
```

Jak łatwo się domyślić, funkcja `h2` tworzy nagłówek HTML drugiego stopnia. Następnie po prostu z każdej pozycji z listy (tej samej, którą wcześniej zapisaliśmy do pliku archiwum) wypisujemy datę, imię użytkownika i wiadomość.

Użytkownicy mogą siedzieć sobie przed komputerami, wpisując cały czas wiadomości i klikając przycisk *Wyślij*. Można w taki sposób zasymulować system elektronicznej „tablicy ogłoszeń”, aby pozwolić użytkownikom oglądać wiadomości innych za każdym razem, gdy wysyłają swoją. W takim przypadku wywoływany jest ten sam skrypt CGI, co oznacza, że poprzednie wartości elementów formularza są automatycznie zachowywane między kolejnymi wywołaniami. Jest to zwłaszcza ważne podczas tworzenia formularzy typu „koszyk na zakupy”.

Kłopoty ze skryptami CGI &&&

Skrypty CGI uruchamiane z serwera WWW działają w zupełnie innym środowisku, niż wywołane z wiersza poleceń. Mimo że zawsze należy sprawdzać poprawność działania skryptu, uruchamiając go w ten drugi sposób¹¹, nawet to nie daje całkowitej gwarancji działania skryptu w jego właściwym środowisku.

Warto przeczytać FAQ *CGI Programming* i jakąś dobrą książkę traktującą o tworzeniu skryptów CGI. Poniżej krótka lista najczęstszych problemów występujących podczas tworzenia skryptów CGI. Prawie wszystkie z nich wywołują ten denerwujący błąd 500 Server Error, który Czytelnik już w niedługim czasie pozna i znienawidzi.

- Jeśli podczas wysyłania do przeglądarki kodu HTML zapomni się o wstawieniu pustego wiersza między nagłówkiem (czyli wierszem `Content-Type`) i treścią, nic nie zadziała. Należy pamiętać o utworzeniu poprawnego wiersza `Content-Type` (i ewentualnie innych nagłówków HTTP) i całkowicie pustego wiersza przed wysłaniem czegokolwiek innego.
- Serwer musi odczytać i wykonać skrypt, więc uprawnienia do niego powinny być zazwyczaj 0555 lub lepiej 0755 (w systemach uniksowych).
- Katalog, w którym skrypt jest umieszczony, musi być wykonywalny, a więc musi mieć uprawnienia przynajmniej 0111 lub lepiej 0755 (w systemach uniksowych).
- Skrypt musi być zainstalowany w odpowiednim katalogu, zdefiniowanym w konfiguracji serwera. W niektórych systemach może to być katalog `/home/httpd/cgi-bin/`.

Istnieje możliwość, że skrypt musi być zakończony jakimś konkretnym rozszerzeniem, np. `.cgi` lub `.pl`. Odradzamy taką konfigurację, polecając jednocześnie wykonywanie skryptów CGI tylko w określonych katalogach, ale za to niezależnie od rozszerzenia. Czasem jest to jednak niemożliwe. Zakładanie z góry, że jakkolwiek plik zakończony przyrostkiem `.cgi` jest wykonywalny, jest niebezpieczne – jeśli jakiś katalog jest zapisywany przez klienty FTP lub zawiera jakieś archiwa (np. cudzych dysków), może to prowadzić do sytuacji, w której na serwerze pojawią się tajemnicze skrypty, wykonywane bez zgody i wiedzy administratora witryny. Oznacza to także, że z serwera nie będzie się dało ściągnąć żadnego pliku, którego nazwa kończy się przyrostkiem `.cgi` lub `.pl`.

Należy pamiętać, że rozszerzenie `.pl` oznacza bibliotekę Perla, a nie kod wykonywalny! Mylenie tych rzeczy bardzo szybko wpędzi programistę w kłopoty¹². Jeśli zachodzi absolutna konieczność użycia przyrostka w celu zdefiniowania skryptu (ponieważ system nie obsługuje na przykład wiersza `#!/usr/bin/perl`), doradzamy przyrostek `.plx`. Lecz nadal mogą występować problemy, o których wspomnieliśmy.

- Serwer wymaga udostępnienia praw do wykonywania skryptu CGI dla konkretnego katalogu. Należy się upewnić, czy dozwolone są zarówno metody `GET`, jak i `POST`. Administrator witryny będzie wiedział, co to oznacza.

¹¹ Więcej o tym jest powiedziane w dokumentacji modułu `CGI.pm` – przyp. red.

¹² Obecnie uwaga ta ma już mniejsze znaczenie: biblioteki na dobre zostały wyparte przez moduły – przyp. red.

- Serwer WWW nie wykonuje skryptu pod UID-em dowolnego użytkownika. Należy się upewnić, że pliki lub katalogi, do których skrypt musi mieć dostęp, są dostępne dla użytkownika, z którego UID działa serwer WWW, czyli np. `nobody`, `wwwuser` lub `httpd`. Może zaistnieć konieczność utworzenia takich plików lub katalogów i udzielenia im praw do zapisu dla każdego. W systemach uniksowych można to zrobić za pomocą polecenia `chmod a+w`, ale zawsze należy pamiętać o niebezpieczeństwach związanych z takimi działaniami.
- Zawsze należy uruchamiać skrypty z opcją Perla `-w`. Pozwala to na otrzymywanie ostrzeżeń o różnych potencjalnych błędach. Te są przekazywane do pliku dziennika błędów serwera. Należy zapytać administratora witryny o ścieżkę do tego pliku i sprawdzać go, gdy wystąpi błąd. Warto także zapoznać się z modulem `CGI::Carp`.
- Należy się upewnić, że wersje i ścieżki do Perla i wszystkich używanych bibliotek (np. `CGI.pm`) są prawidłowe na komputerze, na którym działa serwer WWW.
- Warto włączyć automatyczne opróżnianie bufora dla uchwytu pliku `STDOUT` już na samym początku skryptu, ustawiając wartość zmiennej `$|` na 1. Jeśli używamy modułu `FileHandle` lub któregośkolwiek z modułów IO (np. `IO::File`, `IO::Socket` itd.), możemy zastosować czytelniejszą metodę `autoflush()`:

```
use FileHandle;
STDOUT->autoflush(1);
```
- Należy sprawdzać wartości zwracane przez każde z wywołań systemowych i w przypadku jego niepowodzenia odpowiednio reagować.

Perl i Internet: nie tylko skrypty CGI

Perl doskonale nadaje się także do innych zadań, na przykład do analizy plików dziennika, zarządzania cookies i hasłami, tworzenia aktywnych obrazków i manipulacji grafiką¹³. A to i tak tylko czubek ogromnej góry lodowej możliwości Perla.

Własny system publikacji

Komercyjne systemy publikowania w sieci upraszczają już wiele rzeczy, zwłaszcza dla osób nie potrafiących programować. Nie są jednak aż tak elastyczne, jak prawdziwe języki programowania. Bez dostępu do kodu źródłowego, użytkownik jest zawsze ograniczany przez czyjeś przyzwyczajenia i decyzje: jeśli coś nie działa dokładnie w taki sposób, jak chce, nie może tego zmienić. Niezależnie, ile wspaniałych programów będzie dostępnych dla klienta, programista zawsze będzie potrzebny do wykonania różnych specjalnych zadań, które nie zostały wcześniej przewidziane. Poza tym ktoś musi wcześniej to oprogramowanie napisać.

Perl jest doskonałym narzędziem do tworzenia własnych systemów publikacji – można za jego pomocą w prosty sposób masowo konwertować zwykłe dane do postaci stron HTML. W całym Internecie Perl jest szeroko stosowany do tworzenia i uaktualniania witryn. *The Perl Journal* (<http://www.tpj.com/>) używa Perla do tworzenia wszystkich swych stron. *The Perl Language HomePage* (<http://www.perl.com/>) zawiera prawie dziesięć tysięcy stron automatycznie uaktualnianych przez różne programy w Perlu.

¹³ Warto zapoznać się z modulem `GD.pm`, będącym interfejsem do biblioteki `gd` autorstwa Thomasa Boutella – przyp. red.

Embedded Perl

Najszybszy, najtańszy (bo bezpłatny) i najbardziej popularny serwer WWW Apache może być uruchamiany z Perlem zawartym wewnątrz, przy użyciu modułu `mod_perl`, dostępnego z CPAN. Dzięki temu Perl zostaje w pewien sposób wbudowany w sam serwer i może obsługiwać żądania autoryzacji, reagować na błędy i robić prawie wszystko. Nie wymaga to utworzenia nowego procesu, gdyż Perl jest wbudowany w program serwera. Jeszcze dziwniejsze jest to, że uruchomienie skryptu też nie tworzy procesu – zamiast tego prekompilowany skrypt zostaje uruchomiony w nowym wątku, co znacznie przyspiesza jego wykonanie. Zazwyczaj to właśnie procedura `fork/exec` znacząco spowalnia wykonanie, a nie skrypt sam w sobie.

Innym sposobem przyspieszenia wykonywania skryptów CGI jest użycie modułu `CGI::Fast`. To rozwiązanie nie wymaga już obecności serwera Apache. Więcej informacji znajduje się w podręczniku tego modułu.

Jeśli korzystamy z serwera w systemie WindowsNT, warto zobaczyć stronę ActiveWare (<http://www.activeware.com/>). Są tam nie tylko skompilowane wersje Perla dla platform Windows¹⁴ lecz także PerlScript i PerlIS. PerlScript to język skryptowy ActiveX, pozwalający włączyć Perla w kod strony, tak samo, jak w przypadku JavaScript lub VBScript. PerlIS to ISAPI DLL, pozwalający uruchamiać skrypty Perla bezpośrednio z IIS i innych zgodnych z ISAPI serwerów WWW, co daje znaczny wzrost prędkości.

Automatyzacja za pomocą modułu LWP

Często zachodzi potrzeba sprawdzenia, czy na przykład na stronie WWW nie znajdują się nieaktywne odnośniki; albo które z odnośników były aktualizowane od ostatniego czwartku. Zdarza się też, że musimy ściągnąć obrazki zawarte w dokumencie bądź wykonać lokalną kopię katalogu z dokumentami. A co w przypadku, gdy musimy przedostać się przez serwer proxy lub serwer przekierowujący? Teraz łatwo można to zrobić za pomocą przeglądarki. Ale graficzne interfejsy są niezbyt przydatne do automatyzacji programistycznej, będzie to zatem wolny i mozolny proces, wymagający od użytkownika cierpliwości i większej pracowitości niż ta, na którą większość z nas potrafi się zdobyć¹⁵.

Moduły LWP („Library for WWW access in Perl”) robią to wszystko za użytkownika. Na przykład, ściągnięcie dokumentu z Internetu za pomocą tych modułów jest tak proste, że można je zapisać w jednym wierszu. Na przykład, aby ściągnąć dokument `/perl/index.html` ze strony `www.perl.com`, wystarczy tylko to wpisać w wierszu poleceń:

```
perl -MLWP::Simple -e "getprint 'http://www.perl.com/perl/index.html' "
```

Poza modułem `LWP::Simple`, większość z modułów zawartych w pakiecie LWP jest zorientowanych obiektowo. Oto przykładowy mały programik pobierający jako argumenty adresy URL i wypisujący ich tytuły:

¹⁴ Poczynając od wersji 5.004, standardową dystrybucję Perla można skompilować w systemie Windows – przyp. red.

¹⁵ Według Larry’ego Walla programistę można rozpoznać po trzech cechach: lenistwie, niecierpliwości i pysze – przyp. red.

```
#!/usr/bin/perl
use LWP;
$przegladarka = LWP::UserAgent->new(); # utworz #wirtualna przegladarke
$przegladarka->agent("Mothra/126-Palladium"); # nadaj #nazwe
foreach $url (@ARGV) { # URL-e jako argumenty
    # wykonaj zadanie GET
    $dokument = $przegladarka->request(HTTP::Request->new(GET => $url));
    if ($dokument->is_success) { # znaleziono
        print STDOUT "$url: ", $dokument->title, "\n";
    } else { # cos sie nie udalo
        print STDERR "$0: Nie moge sciagnac $url\n";
    }
}
}
```

Teraz wyraźnie widać, że znajomość obiektów Perla to bardzo ważna rzecz. Tak samo, jak w przypadku CGI.pm, moduły LWP ukrywają większość swych struktur.

Ten skrypt działa w taki sposób: najpierw tworzy obiekt UserAgent, będący czymś w rodzaju zautomatyzowanej wirtualnej przeglądarki. Ten obiekt służy do wysyłania żądań do zdalnych serwerów. Następnie skrypt nadaje obiektowi jakąś bezsensowną nazwę, tylko po to, żeby administrator serwera miał więcej uciechy przy czytaniu dzienników. Następnie ściąga dokument za pomocą żądania GET i jeśli to się powiedzie, wypisuje URL i tytuł strony. W innym przypadku zgłasza błąd.

Oto program wypisujący posortowaną listę łączy i obrazków (z usuniętymi duplikatami) zawartych w adresach URL przekazanych jako argumenty z wiersza poleceń:

```
#!/usr/bin/perl -w
use strict;
use LWP 5.000;
use URI::URL;
use HTML::LinkExor;
my($url,$przegladarka,%widziane);
$przegladarka = LWP::UserAgent->new(); # utworz #'przegladarke'
foreach $url (@ARGV) {
    # pobierz dokument
    my $dokument = $przegladarka->request(HTTP::Request->new(GET=>$url));
    next unless $dokument->is_success;
    next unless $dokument->content_type eq 'text/html';
    # nie bedziemy przetwarzac plikow .gif

    my $podstawa = $dokument->podstawa;

    # teraz wydobadz wszystkie lacza <A...> i <IMG...>
    foreach (HTML::LinkExor->new->parse($dokument->content)->eof->links) {
        my($tag, %lacza) = @$_;
        next unless $tag eq "a" or $tag eq "img";
        my $lacze;
        foreach $lacze (values %lacza) {
            $widziane{ url($lacze,$podstawa)->abs->as_string}++;
        }
    }
}
print join("\n", sort keys %widziane, "\n");
```

Wygląda to trochę skomplikowanie, ale większość trudności leży w zrozumieniu, w jaki sposób poszczególne obiekty i ich metody działają. Nie zamierzamy wyjaśniać ich tutaj, gdyż ta książka i tak jest już zbyt długa. Na szczęście, LWP jest dostarczane razem z wyczerpującą dokumentacją i przykładami.

Lektury

O modułach, referencjach, obiektach i programowaniu Sieci można powiedzieć znacznie więcej, niż udało się nam w tym krótkim rozdziale. Na te tematy można napisać całą książkę, co już uczyniło wielu autorów. Aby nadal zgłębiać wiedzę, warto zatem przeczytać następujące pozycje:

- dokumentację modułu CGI.pm
- bibliotekę LWP dostępną na CPAN.
- Shishir Gundavaram: *CGI Programming on The World Wide Web*, wydawnictwa O'Reilly&Associates
- Clinton Wong: *Web Client Programming With Perl*, wydawnictwa O'Reilly&Associates
- Chuck Musicano i Bill Kennedy: *HTML: The Definitive Guide, Second Edition*, wydawnictwa O'Reilly&Associates
- Lincoln Stein. M.D., Ph.D.: *How to Setup and Maintain a Web Site*, wydawnictwa Addison-Wesley
- Thomas Boutell: *CGI Programming in C and Perl*, wydawnictwa Addison-Wesley
- CGI FAQ, autorstwa Nicka Kew
- podręczniki systemowe perltoot, perlref, perlmod, perlobj

Ćwiczenia

1. Napisz formularz zawierający dwa pola wprowadzania tekstu, których zawartość jest łączona razem po wysłaniu formularza.
2. Napisz skrypt CGI wykrywający typ przeglądarki i wysyłający zależną od tego odpowiedź. (Wskazówka: zmienna środowiskowa `HTTP_USER_AGENT`).